



MADHA
Expertise | Empathy | Excellence
ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**COMMON FOR: DEPARTMENT OF INFORMATION
TECHNOLOGY**

CS8451 – DESIGN AND ANALYSIS OF ALGORITHMS

YEAR / SEM : II / III

R – 2017

LECTURE NOTES

CS8451- DESIGN AND ANALYSIS OF ALGORITHMS

SYLLABUS

- UNIT I INTRODUCTION 9**
Notion of an Algorithm – Fundamentals of Algorithmic Problem Solving – Important Problem Types – Fundamentals of the Analysis of Algorithmic Efficiency – Asymptotic Notations and their properties. Analysis Framework – Empirical analysis - Mathematical analysis for Recursive and Non-recursive algorithms – Visualization
- UNIT II FORCE AND DIVIDE-AND-CONQUER 9**
Brute Force – Computing an – String Matching - Closest-Pair and Convex-Hull Problems - Exhaustive Search - Travelling Salesman Problem - Knapsack Problem - Assignment problem. Divide and Conquer Methodology – Binary Search – Merge sort – Quick sort – Heap Sort - Multiplication of Large Integers – Closest-Pair and Convex - Hull Problems.
- UNIT III DYNAMIC PROGRAMMING AND GREEDY TECHNIQUE 9**
Dynamic programming – Principle of optimality - Coin changing problem, Computing a Binomial Coefficient – Floyd's algorithm – Multi stage graph - Optimal Binary Search Trees – Knapsack Problem and Memory functions. Greedy Technique – Container loading problem - Prim's algorithm and Kruskal's Algorithm – 0/1 Knapsack problem, Optimal Merge pattern - Huffman Trees.
- UNIT IV ITERATIVE IMPROVEMENT 9**
The Simplex Method - The Maximum-Flow Problem – Maximum Matching in Bipartite Graphs, Stable marriage Problem.
- UNIT V COPING WITH THE LIMITATIONS OF ALGORITHM POWER 9**
Lower - Bound Arguments - P, NP NP- Complete and NP Hard Problems. Backtracking – n-Queen problem - Hamiltonian Circuit Problem – Subset Sum Problem. Branch and Bound – LIFO Search and FIFO search - Assignment problem – Knapsack Problem – Travelling Salesman Problem - Approximation Algorithms for NP-Hard Problems – Travelling Salesman problem – Knapsack problem.

TOTAL: 45 PERIODS

TEXT BOOKS:

1. Anany Levitin, —Introduction to the Design and Analysis of Algorithms||, Third Edition, Pearson Education, 2012.
2. Ellis Horowitz, Sartaj Sahni and Sanguthevar Rajasekaran, Computer Algorithms/ C++, Second Edition, Universities Press, 2007.

REFERENCES:

1. Thomas H.Cormen, Charles E.Leiserson, Ronald L. Rivest and Clifford Stein, —Introduction to Algorithms||, Third Edition, PHI Learning Private Limited, 2012.
2. Alfred V. Aho, John E. Hopcroft and Jeffrey D. Ullman, —Data Structures and Algorithms||, Pearson Education, Reprint 2006.
3. Harsh Bhasin, —Algorithms Design and Analysis||, Oxford university press, 2016.
4. S. Sridhar, —Design and Analysis of Algorithms||, Oxford university press, 2014.
5. <http://nptel.ac.in/>

UNIT I

PART A

1. What is an algorithm? Or Define an algorithm. (Apr\May- 2017) Or Define algorithm with its properties.(April/May 2021)

- An algorithm is a finite set of instructions that, if followed, accomplishes a particular task.
- In addition, all algorithms must satisfy the following criteria:
 - input
 - Output
 - Definiteness
 - Finiteness
 - Effectiveness.

2. Define Program.

A program is the expression of an algorithm in a programming language.

3. What is performance measurement?

Performance measurement is concerned with obtaining the space and the time requirements of a particular algorithm.

4. Write the For LOOP general format.

The general form of a for Loop is

```
For variable := value 1 to value 2
Step do
{
    <statement 1>
    <statement n >
}
```

5. What is recursive algorithm?

- ✓ Recursive algorithm makes more than a single call to itself is known as recursive call.
- ✓ An algorithm that calls itself is Direct recursive.
- ✓ Algorithm A is said to be indeed recursive if it calls another algorithm, which in turn calls A

6. What is space complexity?

The space complexity of an algorithm is the amount of memory it needs to run to completion.

7. What is time complexity?

The time complexity of an algorithm is the amount of time it needs to run to completion.

8. Give the two major phases of performance evaluation.

Performance evaluation can be loosely divided into two major phases:

- a prior estimates (performance analysis)
- a posterior testing (performance measurement)

9. Define input size.

The input size of any instance of a problem is defined to be the number of elements needed to describe that instance.

10. Define best-case step count.

The best-case step count is the minimum number of steps that can be

executed for the given parameters.

11. Define worst-case step count.

The worst-case step count is the maximum number of steps that can be executed for the given parameters.

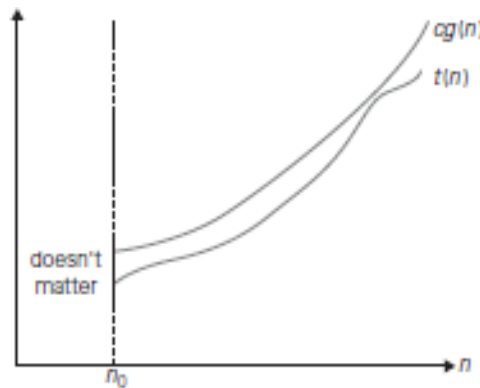
12. Define average step count.

The average step count is the average number of steps executed an instances with the given parameters.

13. Define the asymptotic notation “Big oh” (O)

A function $t(n)$ is said to be in $O(g(n))$ ($t(n) \in O(g(n))$), if $t(n)$ is bounded above by constant multiple of $g(n)$ for all values of n , and if there exist a positive constant c and non negative integer n_0 such that

$$t(n) \leq c * g(n) \quad \text{for all } n \geq n_0.$$

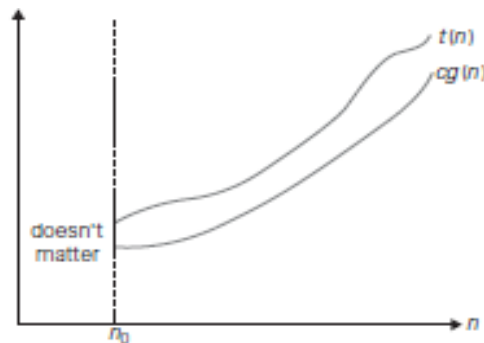


1 Big-oh notation: $t(n) \in O(g(n))$.

14. Define the asymptotic notation “Omega” (Ω). NOV/DEC 2021

A function $t(n)$ is said to be in $\Omega(g(n))$ ($t(n) \in \Omega(g(n))$), if $t(n)$ is bounded below by constant multiple of $g(n)$ for all values of n , and if there exist a positive constant c and non negative integer n_0 such that

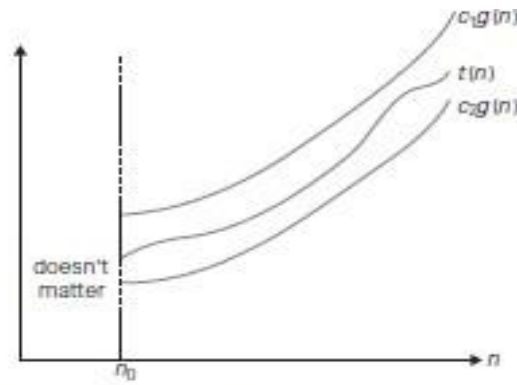
$$t(n) \geq c * g(n) \quad \text{for all } n \geq n_0.$$



Big-omega notation: $t(n) \in \Omega(g(n))$.

15. Define the asymptotic notation “theta” (Θ)

A function $t(n)$ is said to be in $\Theta(g(n))$ ($t(n) \in \Theta(g(n))$), if $t(n)$ is bounded both above and below by constant multiple of $g(n)$ for all values of n , and if there exist a positive constant c_1 and c_2 and non negative integer n_0 such that $c_2 * g(n) \leq t(n) \leq c_1 * g(n)$ for all $n \geq n_0$.



3 Big-theta notation: $t(n) \in \Theta(g(n))$.

16. What is a Computer Algorithm?

An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for any legitimate input in a finite amount of time.

17. What are the features of an algorithm?

More precisely, an algorithm is a method or process to solve a problem satisfying the following properties:

Finiteness-Terminates after a finite number of steps

Definiteness-Each step must be rigorously and unambiguously specified.

Input-Valid inputs must be clearly specified.

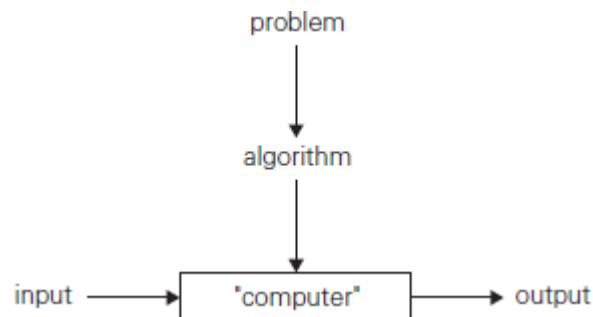
Output-Can be proved to produce the correct output given a valid input.

Effectiveness-Steps must be sufficiently simple and basic.

18. Show the notion of an algorithm.

Dec 2009 / May 2013

An algorithm is a sequence of unambiguous instructions for solving a problem in a finite amount of time.



19. What are different problem types?

- Sorting
- Searching
- String Processing
- Graph problems
- Combinatorial Problems
- Geometric problems
- Numerical problems

20. What are different algorithm design techniques/strategies?

- Brute force
- Divide and conquer
- Decrease and conquer
- Transform and conquer

- Space and time tradeoffs
- Greedy approach
- Dynamic programming
- Backtracking
- Branch and bound

21. How to measure an algorithm's running time? Nov/Dec 2017

Unit for measuring the running time is the algorithms basic operation. The running time is measured by the count of no. of times the basic operations is executed.

Basic operation: the operation that contributes the most to the total running time.

Example: the basic operation is usually the most time-consuming operation in the algorithm's innermost loop.

22. How time efficiency is analyzed?

Let c_{op} – execution time of algorithms basic operation on a particular computer.

$c(n)$ – no. of times this operation need to be executed.

$T(n)$ – running time.

Running time is calculated using the formula

$$T(n) \approx c_{op} c(n)$$

23. What are orders of growth?

Orders of Growth

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

24. What are basic efficiency classes?

Basic Efficiency classes

1	Constant
$\log n$	Logarithmic
n	Linear
$n \log n$	Linearithmic
n^2	Quadratic
n^3	Cubic
2^n	Exponential
$n!$	Factorial

25. Give an example for basic operations.

Input size and basic operation examples

Problem	Input size measure	Basic operation
Searching for key in a list of n items	Number of list's items, i.e. n	Key comparison
Multiplication of two matrices	Matrix dimensions or total number of elements	Multiplication of two numbers

Checking primality of a given integer n	size = number of digits (in binary representation)	Division
Typical graph problem	Number of vertices and/or edges	Visiting a vertex or traversing an edge

26. What are six steps processes in algorithmic problem solving? Dec 2009

1. Understanding the problem.
2. Ascertaining the capabilities of a computational device.
3. Choosing between exact and approximate problem solving.
4. Deciding on appropriate data structures.
5. Algorithm Design Techniques.
6. Methods of specifying an algorithm
7. Proving an algorithm's correctness.
8. Analysing an algorithm.
9. Coding an algorithm.

27. What do you mean by Amortized Analysis?

- ✓ Amortized analysis finds the average running time per operation over a worst case sequence of operations.
- ✓ Amortized analysis differs from average-case performance in that probability is not involved; amortized analysis guarantees the time per operation over worst-case performance.

28. Define order of an algorithm.

Measuring the performance of an algorithm in relation with the input size n is known as order of growth.

29. How is the efficiency of the algorithm defined? Or . How do you measure the efficiency of an algorithm? May/June 2019

The efficiency of an algorithm is defined with the components.

- (i) Time efficiency -indicates how fast the algorithm runs
- (ii) Space efficiency -indicates how much extra memory the algorithm needs

30. What are the characteristics of an algorithm?

Every algorithm should have the following five characteristics

- (i) Input
- (ii) Output
- (iii) Definiteness
- (iv) Effectiveness
- (v) Termination

31. What are the different criteria used to improve the effectiveness of algorithm?

- (i) The effectiveness of algorithm is improved, when the design, satisfies the following constraints to be minimum.
 - Time efficiency - how fast an algorithm in question runs.
 - Space efficiency – an extra space the algorithm requires.
- (ii) The algorithm has to provide result for all valid inputs.

32. Analyse the time complexity of the following segment:

```
for(i=0;i<N;i++)
for(j=N/2;j>0;j--)
sum++;
```

Time Complexity= $N * N/2 = N^2 / 2 \in O(N^2)$

33. Write general plan for analysing non-recursive algorithms.

- i. Decide on parameter indicating an input's size.
- ii. Identify the algorithm's basic operation
- iii. Check the no. of times basic operation executed depends on size of input. if it depends on some additional property, then best, worst, average cases need to be investigated
- iv. Set up sum expressing the no. of times the basic operation is executed. (establishing order of growth)

34. How will you measure input size of algorithms?

The time taken by an algorithm grows with the size of the input. So the running time of the program depends on the size of its input. The input size is measured as the number of items in the input that is a parameter n is indicating the algorithm's input size.

35. Write general plan for analysing recursive algorithms.

- i. Decide on parameter indicating an input's size.
- ii. Identify the algorithm's basic operation
- iii. Checking the no. of times basic operation executed depends on size of input. if it depends on some additional property, then best, worst, average cases need to be investigated
- iv. Set up the recurrence relation, with an appropriate initial condition, for the number of times the basic operation is executed
- v. Solve recurrence (establishing order of growth)

36. What do you mean by Combinatorial Problem?

Combinatorial Problems are problems that ask to find a combinatorial object-such as permutation, a combination, or a subset-that satisfies certain constraints and has some desired properties.

37. Define Little "oh".

The function $f(n) = O(g(n))$ if and only if

$$\lim_{n \rightarrow \infty} f(n) / g(n) = 0$$

38. Define Little Omega.

The function $f(n) = \omega(g(n))$ if and only if

$$\lim_{n \rightarrow \infty} f(n) / g(n) = 0$$

39. Write algorithm using iterative function to find sum of n numbers.

```
Algorithm
sum(a, n)
{
    S := 0.0
    For i=1 to n
    do
        S := S + a[i];
    Return S;
}
```

40. Write an algorithm using Recursive function to find sum of n numbers.

```
Algorithm
Rsum (a, n)
{
    If(n<=0) then
        Return 0.0;
    Else
        Return Rsum(a, n- 1) + a(n);
}
```

41. Describe the recurrence relation for merge sort?

If the time for the merging operation is proportional to n , then the computing time of merge sort is described by the recurrence relation

$$T(n) = \begin{cases} a & n = 1, \\ 2T(n/2) + n & n > 1, \end{cases} \quad \begin{matrix} a \text{ a constant} \\ c \text{ a constant} \end{matrix}$$

42. What is time and space complexity? *Dec 2012 Part A – Refer Q. No. 6 & 7*

43. Define Algorithm validation. *Dec 2012*

The process of measuring the effectiveness of an algorithm before it is coded to know whether the algorithm is correct for every possible input. This process is called validation.

44. Differentiate time complexity from space complexity. *May 2010*

Part A – Refer Q. No. 6 & 7

45. What is a recurrence equation? *May 2010*

A recurrence [relation] is an equation or inequality that describes a function in terms of its values on smaller inputs.

Examples:

Factorial: multiply n by $(n-1)!$

$$T(n) = T(n-1) + O(1) \rightarrow O(n)$$

Fibonacci: add fibonacci($n-1$) and fibonacci($n-2$)

$$T(n) = T(n-1) + T(n-2) \rightarrow O(2^n)$$

46. What do you mean by algorithm? *May 2013 Part A – Refer Q. No. 1, 16 & 18*

47. Define Big Oh Notation. *May 2013 Part A – Refer Q. No. 13*

48. What is average case analysis? *May 2014*

The average case analysis of an algorithm is analysing the algorithm for the average input of size n , for which the algorithm runs at an average between the longest and the fastest time.

49. Define program proving and program verification. *May 2014*

- ✓ Given a program and a formal specification, use formal proof techniques (e.g. induction) to prove that the program behaviour fits the specification.
- ✓ Testing to determine whether a program works as specified.

50. Define asymptotic notation. *May 2014*

Asymptotic notations are mathematical tools to represent time complexity of algorithms for measuring their efficiency.

Types :

- Big Oh notation - 'O'
- Omega notation - 'Ω'
- Theta notation - 'Θ'
- Little Oh notation - 'o'
- Little Omega notation - 'Ω'

51. What do you mean by recursive algorithm? *May 2014 Part A – Refer Q. No. 5*

52. Establish the relation between O and Ω *Dec 2010*

$$f(n) \in \Omega(g(n)) \Leftrightarrow g(n) \in O(f(n))$$

Proof:

$$O(f(n)) = \{g: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c, n_0 \in \mathbb{N} \forall n \geq n_0: g(n) \leq c \cdot f(n)\}$$

$$\Omega(g(n)) = \{f: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c, n_0 \in \mathbb{N} \forall n \geq n_0: f(n) \geq c \cdot g(n)\}$$

$$\text{Step 1/2: } f(n) \in \Omega(g(n)) \Leftrightarrow g(n) \in O(f(n))$$

$\exists c, n_0 \in \mathbb{N} \forall n \geq n_0: f(n) \geq c \cdot g(n) \Rightarrow f(n)g(n) \geq c \Rightarrow 1g(n) \geq cf(n) \Rightarrow g(n) \leq 1c \cdot f(n)$

And this is exactly the definition of $O(f(n))$.

Step 2/2: $f(n) \in \Omega(g(n)) \Leftarrow g(n) \in O(f(n))$

$\exists c, n_0 \in \mathbb{N} \forall n \geq n_0: g(n) \leq c \cdot f(n) \Rightarrow \dots \Rightarrow f(n) \geq 1c \cdot g(n)$

Hence proved.

53. If $f(n) = a_m n^m + \dots + a_1 n + a_0$. Prove that $f(n) = O(n^m)$.

Dec 2010 Refer Class note.

54. What is best case analysis? Or Best case efficiency.

The best case analysis of an algorithm is analysing the algorithm for the best case input of size n , for which the algorithm runs the fastest among all the possible inputs of that size.

55. what do you mean worst case efficiency of algorithm. Nov/Dec 2017

The worst case analysis of an algorithm is analysing the algorithm for the worst case input of size n , for which the algorithm runs the longest among all the possible inputs of that size.

56. Consider an algorithm that finds the number of binary digits in the binary representation of a positive decimal integer. (AU april/may 2015)

Number of major comparisons = $\lfloor \log_2 n \rfloor + 1 \in \log_2 n$.

Algorithm 3: Finding the number of binary digits in the binary representation of a positive decimal integer.

Algorithm Binary(n)

count:=1;

while $n > 1$

do

count:=count+ 1;

$n := \lfloor n/2 \rfloor$;

end

return count;

57. write down the properties of asymptotic notations. (AU april/may 2015)

The following property is useful in analyzing algorithms that comprise two consecutively executed parts.

Theorem

If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$ then,

$t_1(n) + t_2(n) \in O(\max \{g_1(n), g_2(n)\})$

Proof

Since $t_1(n) \in O(g_1(n))$, there exist some constant C_1 and some non negative integer n_1 such that

$t_1(n) \leq C_1 (g_1(n))$ for all $n \geq n_1$

Since

$t_2(n) \in O(g_2(n))$

$t_2(n) \leq C_2 (g_2(n))$ for all $n \geq n_2$

Let us denote,

$C_3 = \max \{C_1, C_2\}$ and

Consider $n \geq \max \{n_1, n_2\}$, so that both the inequalities can be used.

The addition of two inequalities becomes,

$$\begin{aligned} t_1(n) + t_2(n) &\leq C_1 (g_1(n)) + C_2 (g_2(n)) \\ &\leq C_3 (g_1(n)) + C_3 (g_2(n)) \\ &\leq C_3 2 \max \{g_1(n), (g_2(n))\} \end{aligned}$$

Hence,

$t_1(n) + t_2(n) \in O(\max \{g_1(n), g_2(n)\})$,

with the constants C and n_0 required by the definition being $2C_3 = 2 \max \{C_1, C_2\}$ and $\max \{n_1, n_2\}$ respectively.

The property implies that the algorithms overall efficiency will be determined by the part with a larger order of growth.

(i.e.) its least efficient part is

$$\begin{aligned} t_1(n) \in O(g_1(n)) & \quad t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\}) \\ t_2(n) \in O(g_2(n)) & \end{aligned}$$

58. Give the Euclid's algorithm for computing $gcd(m, n)$ (AU nov 2016) or write an algorithm to compute the greatest common divisor of two numbers (Apr/ May-2017)(or)

Give the Euclid's algorithm for computing gcd of two numbers. (May/June 2018)

	2	
Polynomial	Quadratic	Quadratic
1	0	1
2	1	4
4	6	16
8	28	64
10	45	10^z
10^z	4950	10^+
Complexity	Low	High
Growth	Low	high

ALGORITHM *Euclid_gcd(m, n)*

```
//Computes gcd(m, n) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while n ≠ 0 do
    r ← m mod n
    m ← n
    n ← r
return m
```

Example: $gcd(60, 24) = gcd(24, 12) = gcd(12, 0) = 12$.

59. Compare the order of growth $n(n-1)/2$ and n^2 . (AU nov 2016)

$n(n-1)/2$ is lesser than the half of n^2

60. The $(\log n)$ th smallest number of n unsorted numbers can be determined in $O(n)$ average-case time

Ans: True

61. Fibonacci algorithm and its recurrence relation

Algorithm for computing Fibonacci numbers

First method

Algorithm F(n)

//Computes the nth Fibonacci number recursively by using its definition.

//Input: A nonnegative integer n

//Output: The nth Fibonacci number

if $n < 1$

return n

Else

return $F(n-1) + F(n-2)$

the algorithm's basic operation is addition.

Let $A(n)$ is the number of additions performed by the algorithm to compute $F(n)$.

The number of additions needed to compute $F(n-1)$ is $A(n-1)$ and the number of additions needed to compute $F(n-2)$ is $A(n-2)$.

62. Design an algorithm to compute the area and circumference of a circle

1. Find Area, Diameter and Circumference of a Circle.

ALGORITHM

```
Step 1: Start
Step 2: Initialize PI to 0
Step 3: Read radius of the circle
Step 4: Calculate the product of radius with itself and PI value
Step 5: Store the result in variable area
Step 6: Calculate the product of radius with 2
Step 7: Store the result in variable diameter
Step 8: Calculate the product of PI and diameter value
Step 9: Store the result in variable circumference
Step 10: Print the value of area, diameter and circumference
Step 11: Stop
```

PSEUDO CODE

```
Step 1: Begin
Step 2: Set PI to 3.14
Step 3: Read radius
Step 4: Compute the product of radius with itself and PI value
Step 5: Set the result to area
Step 6: Compute the product of radius with 2
Step 7: Set the result to diameter
Step 8: Calculate the product of PI and diameter value
Step 9: Set the result to circumference
Step 10: Display area, diameter, circumference
Step 11: End
```

63. What is a basic operation?

A basic operation could be: An assignment. A comparison between two variables. An arithmetic operation between two variables. The worst-case input is that input assignment for which the most basic operations are performed.

Basic Operations on Sets. The set is the basic structure underlying all of mathematics. In algorithm design, sets are used as the basis of many important abstract data types, and many techniques have been developed for implementing set-based abstract data types.

64. Define algorithm. List the desirable properties of an algorithm.

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

An algorithm must satisfy the following properties: Input: The algorithm must have input values from a specified set. The output values are the solution to a problem. Finiteness: For any input, the algorithm must terminate after a finite number of steps. Definiteness: All steps of the algorithm must be precisely defined.

65. Define best, worst, average case time complexity.

- The *worst-case complexity* of the algorithm is the function defined by the maximum number of steps taken on any instance of size n . It represents the curve passing through the highest point of each column.
- The *best-case complexity* of the algorithm is the function defined by the minimum number of steps taken on any instance of size n . It represents the curve passing through the lowest point of each column.
- Finally, the *average-case complexity* of the algorithm is the function defined by the average number of steps taken on any instance of size n .

66. Prove that the of $f(n)=o(g(n))$ and $g(n)=o(f(n))$, then $f(n)=\theta g(n)$. OR state the transpose symmetry property of O and Ω

April/May 2019, Nov/Dec

2019

Given function:

$f(n)$ and $g(n)$

$f(n) = O(g(n))$ when $f(n) \leq C_1 g(n)$ for all $n \geq n_0$ (1)

$f(n) = \Omega(g(n))$ when $f(n) \geq C_2 g(n)$ for all $n \geq n_0$ (2)

from (1) and (2)

$C_2 g(n) \leq f(n) \leq C_1 g(n)$ for all $n \geq n_0$ (3)

(i.e) $\Theta(g(n)) = O(g(n))\Omega(g(n))$

From (3) $f(n) = \Theta(g(n))$ hence proved

67. Define recursion

A function may be recursively defined in terms of itself. A familiar example is the Fibonacci number sequence: $F(n) = F(n - 1) + F(n - 2)$.

For such a definition to be useful, it must be reducible to non-recursively defined values: in this case $F(0) = 0$ and $F(1) = 1$.

ccurs when a thing is defined in terms of itself or of its type.

Recursion is used in a variety of disciplines ranging from linguistics to logic.

The most common application of recursion is in mathematics and computer science, where

a function being defined is applied within its own definition.

While this apparently defines an infinite number of instances (function values), it is often

done in such a way that no loop or infinite chain of references can occur.

68. List the reasons for choosing an approximate algorithm.

Approximation algorithms are typically used **when finding an optimal solution is intractable**, but can also be used in some situations where a near-optimal solution can be found quickly and an exact solution is not needed. Many problems that are NP-hard are also non-approximable assuming $P \neq NP$.

1. Explain the notion of an algorithm with diagram.

May2014

Synopsis:

- Introduction
- Definition
- Diagram
- Characteristics of an Algorithm / Features of an Algorithm
- Rules for writing an Algorithm
- Implementation of an Algorithm
- Order of an Algorithm
- Program
- Example : GCD

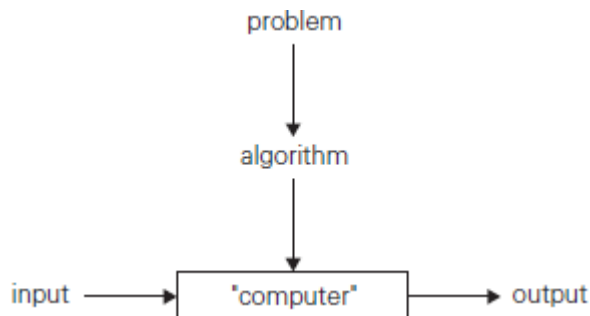
Introduction:

- An algorithm is a sequence of finite number of steps involved to solve a particular problem.
- An input to an algorithm specifies an instance of the problem the algorithm solves.
- An algorithm can be specified in a natural language or in a pseudo code.
- Algorithm can be implemented as computer programs.
- The same algorithm can be represented in several different ways.
- Several algorithms for solving the same problem may exist.
- Algorithms for the same problem can be based on different ideas and can solve the problem with dramatically different speeds.

Definition:

- An algorithm is a sequence of non ambiguous instructions for solving a problem in a finite amount of time.
- Each algorithm is a module, designed to handle specific problem.
- The non ambiguity requirement for each step of an algorithm cannot be compromised.
- The range of inputs for which an algorithm works has to be specified carefully.

Diagram:



Characteristics of an algorithm / Features of an Algorithm

The important and prime characteristics of an algorithm are,

- ✓ **Input:**Zero or more quantities are externally supplied.
- ✓ **Output:**At least one quantity is produced.
- ✓ **Definiteness:**Each instruction is clear and unambiguous.
- ✓ **Finiteness:**For all cases the algorithm terminates after a finite number of steps.
- ✓ **Efficiency:**Every instruction must be very basic.
- ✓ An algorithm must be expressed in a fashion that is completely free of ambiguity.
- ✓ It should be efficient.
- ✓ Algorithms should be concise and compact to facilitate verification of their correctness.

Writing an algorithm

- Algorithm is basically a sequence of instructions written in simple English language.

- The algorithm is broadly divided into two sections

Algorithm heading

It consists of name of algorithm, problem description, input and output.

Algorithm Body

It consists of logical body of the algorithm by making use of various programming constructs and assignment statement.

Rules for writing an algorithm.

Algorithm is a product consisting of heading and body. The heading consists of keyword **algorithm** and name of the algorithm and parameter list. The syntax is

Algorithm name (p1, p2,.....pn)

1. Then in the heading section we should write following things :
 - // Problem Description;**
 - // Input:**
 - //Output:**
2. Then body of an algorithm is written, in which various programming constructs like if, for, while or some assignment statement may be written.
3. The compound statements should be enclosed within { and } brackets.
4. Single line comments are written using // as beginning of comment.
5. The **identifier** should begin by letter and not by digit. An identifier can be a combination of alphanumeric string.
 - It is not necessary to write data types explicitly for identifiers. It will be represented by the context itself.
 - Basic data types used are integer, float, and char, Boolean and so on.
 - The pointer type is also used to point memory locations.
 - The compound data type such as structure or record can also be used.
6. Using assignment operator ← an assignment statement can be given.
For instance: **Variable ← expression**
7. There are other types of operators' such as Boolean operators such as true or false. Logical operators such as AND, OR, NOT. And relational operators such as <, <=, >, >=, =, !=.
8. The array indices are stored with in square brackets '[' '']. The index of array usually starts at zero. The multidimensional arrays can also be used in algorithm.
9. The inputting and outputting can be done using read and write.
For example:
Write ("this message will be displayed on console ");
Read (Val);
10. The conditional statements such as if –then – else are written in following form
If (condition) then statement
If (condition) then statement else statement
If the **if – then** statement is of compound type then {and} should be used for enclosing block
11. While statement can be written as :
While (condition)do
{
Statement 1
Statement 2
:

Statement n

}

While the condition is true the block enclosed with { } gets executed otherwise statement after } will be executed.

12. The general form for writing for loop is :

For variable ← value₁ to value_n do

{

Statement 1

Statement 2

:

Statement n

}

Here value₁ is initialization condition and value_n is a terminating condition the step indicates the increments or decrements in value₁ for executing the for loop.

Sometime a keyword step is used to denote increment or decrement the value of variable for example

For i ← 1 to n step 1

{

Write (i)

}



Here variable i is incremented by 1 at each iteration

13. The **repeat – until** statement can be written as

Repeat

Statement 1

Statement 2

:

Statement n

Until (condition)

14. The **break** statement is used to exit from inner loop. The return statement is used to return control from one point to another. Generally used while exiting from function

Note: The statements in an algorithm executes in sequential order i.e. in the same order as they appear – one after the other

Example 1 : Write an algorithm to count the sum of n numbers

Algorithm sum (1, n)

//Problem description : this algorithm is for finding the

//sum of given n numbers

//Input: 1 to n numbers

//Output: the sum of n numbers

Result ← 0

For i 1 to n do

 i ← i+1

 Result ← result + i

Return result

Example 2: Write an algorithm to check whether given number is even or odd.

Algorithm eventest (val)

//Problem description : this algorithm test whether given

//number is even or odd

//Input: the number to be tested i.e .val

//Output: appropriate messages indicating even or odd


```

If (val % 2 = 0) then
    Write (“given number is even “)
Else
    Write (“given number is odd”)

```

Example 3: Write an algorithm for sorting the elements.

```

Algorithm sort (a, n)
    //Problem description: sorting the elements in ascending
    //order
    //Input: an array in which the elements in ascending order
    //is total number of elements in the array
    //Output: the sorted array
    For i 1 to n do
        For j i + 1 to n-1 do
            If (a[i]>a[j]) then
                {
                    temp ← a[i]
                    a[i] ←a[j]
                    a[j] ←temp
                }
    Write ( “ list is sorted “)

```

Example 4: Write an algorithm to find factorial of n number.

```

Algorithm fact (n)
    //Problem description: this algorithm finds the factorial.
    //for given number n
    //Input : the number n of which the factorial is to be
    //calculated.
    //Output : factorial value of given n number.
    If( n ← 1) then
        Return 1
    Else
        Return n * fact(n-1)

```

Example 5:

Write an algorithm to perform multiplication of two matrices

```

Algorithm mul (A, b, n)
    //Problem description: this algorithm is for computing
    //multiplication of two matrices
    //Input : the two matrices A, B and order of them as n
    //Output : The multiplication result will be in matrix c
    For i ← 1 to n do
        For j ← 1 to n do
            C [i,j] ← 0

            For k ← 1 to n do
                C[I ,j ] ←c[i, j] +A[i,k]B[k,j]

```

Implementation of algorithms

An algorithm describes what the program is going to perform. It states some of the actions to be executed and the order in which these actions are to be executed.

The various steps in developing algorithm are,

1. Finding a method for solving a problem. Every step of an algorithm should be in a precise and in a clear manner. Pseudo code is also used to describe the algorithm.
2. The next step is to validate the algorithm. This step includes, all the algorithm should be done manually by giving the required input, performs the required steps including in the algorithm and should get the required amount of output in an finite amount of time.
3. Finally, implement the algorithm in terms of programming language.

Order of an algorithm

The order of an algorithm is a standard notation of an algorithm that has been developed to represent function that bound the computing time for algorithms. It is an order notation. It is usually referred as O-notation.

Example

Problem size = 'n'

Algorithm = 'a' for problem size n

The document mechanism execution = Cn^2 times

where C – constant

Then the order of the algorithm 'a' = $O(n^2)$

where n^2 = Complexity of the algorithm 'a'.

Program

- A set of explicit and unambiguous instructions expressed using a programming languages constructs is called a program.
- An algorithm can be converted into a program, using any programming language. Pascal, Fortran, COBOL, C and C++ are some of the programming languages.

Difference between program and algorithm:

Sno	Algorithm	Program
1	Algorithm is finite.	Program need to be finite.
2	Algorithm is written using natural language or algorithmic language.	Programs are written using a specific programming language.

1.A. write an algorithm using recursion that determines the GCD of two numbers. Determine the time and space complexity Nov/Dec 2019

Example : Calculating Greatest common Divisor

The Greatest common Divisor (GCD) of two non zero numbers a and b is basically the largest integer that divides both a and b evenly i.e with a remainder of zero.

GCD using three methods

1. Euclid's algorithm
2. Consecutive integer checking algorithm
3. Finding Gusing repetitive factors

Euclid's algorithm to compute Greatest Common Divisor (GCD) of two non negative integers.

Euclid's algorithm is based on applying related equality

$$\text{gcd}(m, n) = \text{gcd}(n, m \bmod n) \text{ until the } m \text{ and } n \text{ is equal to } 0$$

Where $m \bmod n$ is the remainder of the division of m by n

Step 1: Start

Step 2: If $n = 0$, return the value of m as the answer and stop, otherwise proceed to step 3.

Step 3: Divide m by n and assign the value of the remainder to r .
 Step 4: Assign the value of n to m and the value of r to n . Goto step 2
 Step 5: Stop

ALGORITHM *Euclid(m, n)*

```
//Computes gcd(m, n) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while n ≠ 0 do
  r ← m mod n
  m ← n
  n ← r
return m
```

Example, gcd (60,24) can be computed as follows,

gcd (60,24)	gcd (m, n)
	m =60, n=24;
	m/n = 2 (remainder 12)
	n=m=24
	r=n=12
gcd (24, 12)	m/2 = 2 (remainder 0)
	n=m=12
	r=n=0

gcd (12, 0) =12

Hence, gcd(60, 24) = gcd(24,12)=gcd(12,0)=12

2. Consecutive integer checking algorithm

In this method while finding the GCD of a and b we first of all find the minimum value of them. Suppose if , value of b is minimum then we start checking the divisibility by each integer which is lesser than or equal to b .

Example:

$a = 15$ and $b = 10$ then
 $t = \min(15, 10)$
 since 10 is minimum we will set value of $t = 10$ initially.

Consecutive integer checking algorithm for computing gcd(m, n)

Step 1: Start
 Step 2: Assign the value of $\min\{m, n\}$ to t
 Step 3: Divide m by t . If the remainder of this division is 0, go to step 4,
 Otherwise goto step 5.
 Step 4: Divide n by t . If the remainder of this division is 0, return the value
 of t as the answer and stop. Otherwise proceed to step 5.
 Step 5: Decrease the value of t by 1. Go to step 3.
 Step 6: Stop

Algorithm GCD intcheck (a,b)

```
//Problem description : this algorithm computes the GCD of //two
//numbers a and b using consecutive integer checking
//method
//Input : two integers a and b
//Output: GCD value of a and b
t ← min ( a, b)
while (t>=1) do
{
  If ( a mod t == 0 AND b mod t == 0) then
    Return t
```

```

        Else
            t ← t-1
    }
    Return 1

```

3. Finding GCD using repetitive factors

The third procedure for finding the greatest common divisor is middle school procedure.

Middle School Method

For the numbers 60 and 24

$$\begin{array}{r}
 2 \overline{)60} \\
 \underline{2 \quad 30} \\
 3 \overline{)15} \\
 \underline{3 \quad 5} \\
 5
 \end{array}
 \qquad
 \begin{array}{r}
 2 \overline{)24} \\
 \underline{2 \quad 12} \\
 2 \overline{)6} \\
 \underline{2 \quad 3} \\
 3
 \end{array}$$

$$\begin{aligned}
 60 &= \underline{2 \times 2 \times 3} \times 5 \\
 24 &= \underline{2 \times 2 \times 3} \times 2 \\
 \text{gcd}(60, 24) &= \underline{2 \times 2 \times 3} = 12
 \end{aligned}$$

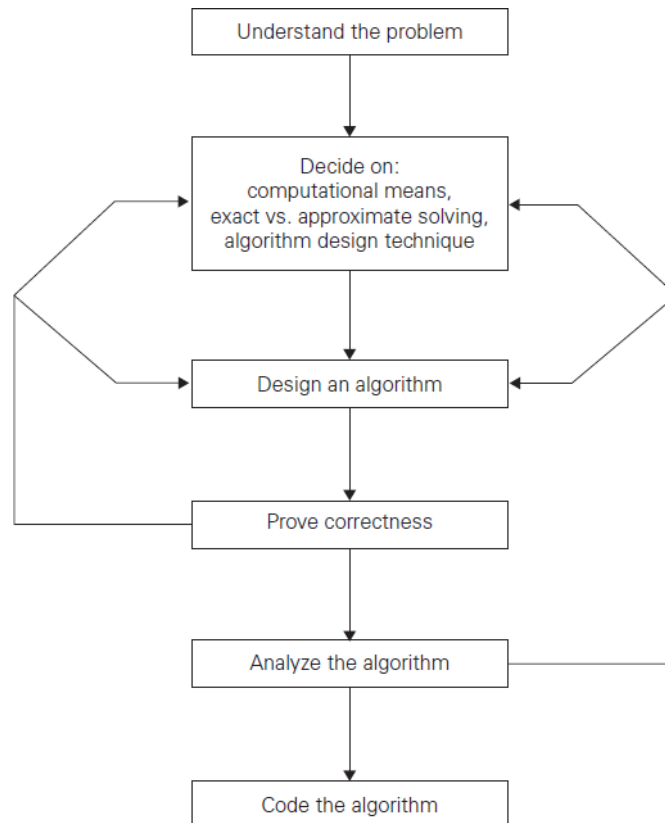
Algorithm:

- Step 1: Start
- Step 2: Find the prime Factor of m.
- Step 3: Find the prime factors of n.
- Step 4: Identify all the common factors in the two prime expressions Found in step 2 and step 3. If P is a common factor occurring pm and pn times in m and n respectively. It should be repeated min (pm, and pn) times.
- Step 5: Compute the product of the all the common factors and return it as the greatest common divisor of the numbers given.
- Step 6: Stop.

2. Explain the Fundamentals of Algorithmic problem solving. Or explain the steps involved in problem solving *May 2014 ,April/May 2019*

Sequential steps in designing and analysing an algorithm

1. Understanding the problem.
2. Ascertaining the capabilities of a computational device.
3. Choosing between exact and approximate problem solving.
4. Deciding on appropriate data structures.
5. Algorithm Design Techniques.
6. Methods of specifying an algorithm
7. Proving an algorithm's correctness.
8. Analysing an algorithm.
9. Coding an algorithm.



1. Understanding the problem:

- ✓ To design an algorithm, understand the problem completely by reading the problem's description carefully.
- ✓ Read the problem description carefully and clear the doubts.
- ✓ Specify exactly the range of inputs the algorithm need to handle.
- ✓ Once the problem is clearly understandable, then determine the overall goals but it should be in a precise manner.
- ✓ Then divide the problem into smaller problems until they become manageable size.

2. Ascertaining the capabilities of a computational device

Sequential Algorithm:

- ✓ Instructions are executed one after another, one operation at a time.
- ✓ This is implemented in RAM model.

Parallel Algorithm:

- ✓ Instructions are executed in parallel or concurrently.

3. Choosing between exact and appropriate problem solving

- ✓ The next principal decision is to choose between solving the problem exactly or solving the problem approximately.
- ✓ The algorithm used to solve the problem exactly called **exact algorithm**.
- ✓ The algorithm used to solve the problem approximately is called **approximation algorithm**.

Reason to choose approximate algorithm

- There are important problems that simply cannot be solved exactly such as
 - Extracting square roots.
 - Solving non linear equations.
 - Evaluating definite integrals.
- ✓ Available algorithms for solving problem exactly can be unacceptably slow, because

of the problem's intrinsic complexity. Ex: Travelling salesman problem

4. Deciding on appropriate data structures

Data structure is important for both design and analysis of algorithms.

Algorithm + Data Structures = Programs.

In Object Oriented Programming, the data structure is important for both design and analysis of algorithms.

The variability in algorithm is due to the data structure in which the data of the program are stored such as

1. How the data are arranged in relation to each other.
2. Which data are kept in memory
3. Which data are kept in files and how the files are arranged.
4. Which data are calculated when needed?

5. Algorithm Design Techniques

An algorithm design techniques or strategy or paradigm is general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.

Uses

- ✓ They provide guidance for designing algorithms or new problems.
- ✓ They provide guidance to problem which has no known satisfied algorithms.
- ✓ Algorithm design technique is used to classify the algorithms based on the design idea.
- ✓ Algorithm design techniques can serve as a natural way to categorize and study the algorithms.

6. Methods of specifying an algorithm

There are two options, which are widely used to specify the algorithms.

They are

- Pseudo code
- Flowchart

Pseudo code

- A pseudo code is a mixture of natural language and programming language constructs.
- A pseudo code is more precise than a natural language
- For simplicity, declaration of the variables is omitted.
- For, if and while statements are used to show the scope of the variables.
- "←" (Arrow) - used for the assignment operation.
- "///" (two slashes) - used for comments.

Flow chart

- It is a method of expressing an algorithm by a collection of connected geometric shapes containing description of the algorithms steps.
- It is very simple algorithm.
- This representation technique is inconvenient.

7. Proving an Algorithm's correctness

Once an algorithm has been specified, then its correctness must be proved.

- ✓ An algorithm must yield a required result for every legitimate input in a finite amount of time.
- ✓ A mathematical induction is a common technique used to prove the correctness of the algorithm.
- ✓ In mathematical induction, an algorithm's iterations provide a natural sequence of steps needed for proofs.
- ✓ If the algorithm is found incorrect, need to redesign it or reconsider other decisions.

8. Analysing an algorithm

- ✓ Efficiency of an algorithm is determined by measuring the time, space and amount of

- resources, it uses for executing the program.
- ✓ The efficiency of the algorithm is determined with respect to central processing units time and internal memory.
- ✓ There are two types of algorithm efficiency.

They are

- Time efficiency (or) Time Complexity
- Space efficiency (or) Space Complexity

Time Efficiency / Time Complexity

- ✓ Time efficiency indicates how fast the algorithm runs.
- ✓ The time taken by a program to complete its task depends on the number of steps in an algorithm.
- ✓ The time required by a program to complete its task will not always be the same.
- ✓ It depends on the type of problem to be solved.

It can be of two types.

- Compilation Time
- Run Time (or) Execution Time
- ✓ The time (T) taken by an algorithm is the sum of the compile time and execution time.

Compilation Time

- ✓ The amount of time taken by the compiler to compile an algorithm is known as compilation time.
- ✓ During compilation time, it does not calculate the executable statements, it calculates only the declaration statements and check for any syntax and semantic errors.
- ✓ The different compilers can take different times to compile the same program.

Execution Time

- ✓ The execution time depends on the size of the algorithm.
- ✓ If the number of instructions in an algorithm is large then the run time is also large.
- ✓ If the number of instructions in an algorithm is small then the time need to execute the program is small.
- ✓ The execution time is calculated for executable statements and not for the declaration statements.
- ✓ The complexity is normally expressed as an order of magnitude.
- ✓ Example: $O(n^2)$
- ✓ The time complexity of a given algorithm is defined as computation of function $f()$ as a total number of statements that are executed for computing the value $f(n)$.
- ✓ The time complexity is a function which depends on the value of n .

The time complexity can be classified as 3 types.

They are

1. Worst Case analysis
2. Average Case analysis
3. Best Case analysis

Worst Case Analysis

- ✓ The worst case complexity for a given size corresponds to the maximum complexity encountered among all problem of the same size.
- ✓ Worst case complexity takes a longer time to produce a desired result.

This can be represented by a function $f(n)$.

$$f(n) = n^2 \text{ or } n \log n$$

Average Case Analysis

- ✓ The average case analysis is also known as the expected complexity which gives measure of the behaviour of an algorithm averaged over all possible problem of the same size.
- ✓ Average case is the average time taken by an algorithm for producing a desired output.

Best Case Analysis

- ✓ Best case is a shortest time taken by an algorithm to produce the desired result.

Space Complexity

- ✓ Space efficiency indicates how much extra memory the algorithm needs.
- ✓ The amount of storage space taken by the algorithm depends on the type of the problem to be solved.
- ✓ The space can be calculated as,
- ✓ A fixed amount of memory occupied by the space for the program code is space occupied by the variable used in the program.
- ✓ A variable amount of memory occupied by the component variable dependent on the problem is being solved.
- ✓ This space is more or less depending upon whether the program uses iterative or recursive procedures.

There are three different space considered for determining the amount of memory used by the algorithm.

They are

- Instruction Space
- Data Space
- Environment Space

Instruction Space

- ✓ When the program gets compiled, then the space needed to store the compiled instruction in the memory is called instruction space.
- ✓ The instruction space independent of the size of the problem

Data Space

- ✓ The memory space used to hold the variables of data elements are called data space.
- ✓ The data space is related to the size of the Problem

Environment Space

- ✓ It is the space in memory used only on the execution time for each Function call.
- ✓ It maintains runtime stack in that it holds returning address of the previous functions.
- ✓ Every function on the stack has return value and a pointer on it.

Characteristics of an algorithms

- Simplicity
- Generality

Simplicity

- Simpler algorithms are easier to understand.
- Simpler algorithms are easier to program.
- The resulting programs contains only few bugs.
- Simpler algorithms are more efficient compared to the complicated alternatives.

Generality

- The characteristic of an algorithm generality has two issues.
- They are
 - Generality' of the problem the algorithm solves.
 - Range of inputs it accepts.

9. Coding an Algorithm

- ✓ Implementing an algorithm correctly is necessary but not sufficient to diminish the algorithm's power by an inefficient implementation.
- ✓ The standard tricks such as computing a loop's invariant (an expression that does not change its value) outside the loop, collecting common sub expressions, replacing expensive operations by cheaper ones and so on should be known to the programmers such factors can speed up a program only by a constant factor, where as a better algorithm can make a difference in running time by orders of magnitude.

- ✓ Once an algorithm has been selected, a 10-50% speed up may be worth an effort.
- ✓ An algorithm's optimality is not about the efficiency of an algorithm but about the complexity of the problem it solves.

3. Explain the important problem types.

Some of the most important problem types are

1. Sorting
2. Searching
3. String Matching (or) String processing
4. Graph Problems
5. Combinatorial problems
6. Geometric problems
7. Numerical Problems

1. Sorting

- ✓ Sorting means arranging the elements in increasing order or in decreasing order.
- ✓ The sorting can be done on numbers, characters (alphabets), string or employees record.
- ✓ Many algorithms are used to perform the task of sorting.
- ✓ Sorting is the operation of arranging the records of a table according to the key value of the each record.
- ✓ A table of a file is an ordered sequence of records $r[1], r[2].. r[n]$ each containing a key $k[1], k[2]... k[n]$. The table is sorted based on the key.

Properties of Sorting Algorithms

The two properties of Sorting Algorithms are

1. Stable
2. In-place

Stable:

- ✓ A sorting algorithm is called **stable**, if it preserves the relative order of any two equal elements in its input.
- ✓ In other words, if an input list contain two equal elements in positions i and j , where $i < j$, then in the sorted list they have to be in position i' and j' respectively, such that $i' < j'$

In-place

- ✓ An algorithm is said to be **in-place** if it does not require extra memory, except, possibly for a few memory units.

The important **criteria for the selection of a sorting** method for the given set of data items are as follows.

1. Programming time of the sorting algorithm.
2. Execution time of the program
3. Memory space needed for the programming environment

The **main objectives** involved in the design of sorting algorithms are

1. Minimum number of exchanges.
2. Large volume of data blocks movement.

Types of Sorting

The two major classification of sorting methods are

1. Internal Sorting methods
2. External Sorting methods

Internal Sorting

- ✓ The key principle of internal sorting is that all the data items to be sorted are retained in the main memory and random access memory.
- ✓ This memory space can be effectively used to sort the data items.
- ✓ The various internal sorting methods are
 1. Bubble sort
 2. Selection sort
 3. Shell sort

4. Insertion sort
5. Quick sort
6. Heap sort

External Sorting

- ✓ The idea behind the external sorting is to move data from secondary storage to main memory in large blocks for ordering the data.
- ✓ The most commonly used external sorting method is merge sort.

2. Searching

- ✓ One of the important applications of array is searching
- ✓ Searching is an activity by which we can find out the desired element from the list. The element which is to be searched is called **search key**
- ✓ There are many searching algorithm such as sequential search , Fibonacci search and more.

Searching in dynamic set of elements

- ✓ There may be of elements in which repeated addition or deletion of elements occur.
- ✓ In such a situation searching an element is difficult.
- ✓ To handle such lists supporting data structures and algorithms are needed to make the list balanced (organized)

3. String processing

A **string is a collection** of characters from an alphabet.

Different type of strings are

- Text string
- Bit string

Text String It is a collection of letters, numbers and special characters.

Bit String It is collection of zeros and ones.

- Operations performed on a string are
 1. Reading and writing strings
 2. String concatenation
 3. Finding string length
 4. String copy
 5. String comparison
 6. Substring operations
 7. Insertions into a string
 8. Deletions from a string
 9. Pattern matching

Pattern Matching or String matching

The process of searching for an occurrence of word in a text is called Pattern matching.

Some of the algorithms used for pattern matching are

1. Simple pattern matching algorithm
2. Pattern matching using Morris Pratt algorithm
3. Pattern matching using Knuth-Morris-Pratt algorithm

4. Graph Problems

- ✓ Graph is a collection of vertices and edges.
- ✓ Formally, a graph $G = \{ V, E \}$ is defined by a pair of two sets.
- ✓ A finite set V of items called Vertices and a set E of pairs of these items called edges.
- ✓ If the pairs of vertices are ordered, then G is called a directed graph because every edge is directed.
- ✓ In a directed graph the direction between two nodes are not same $G(V,W) \neq G(W,V)$
- ✓ If the pair of the vertices are unordered then G is called an undirected graph.
- ✓ In undirected graph, the edges has no specific direction.
- ✓ The graph problems involve graph traversal algorithms, shortest path algorithm and topological sorting and so on. Some graph problems are very hard to solve.
- ✓ For example travelling salesman problem, graph colouring problems

5. Combinatorial Problems

- ✓ The travelling salesman problem and the graph colouring problems are examples of combinatorial problems.
- ✓ A combinatorial object such as a permutation a combination or a subset that satisfies certain constraints and has some desired property such as maximizes a value or minimizes a cost should be find.
- ✓ Combinatorial problems are the **most difficult problems**.

The reason is,

1. As problem size grows the combinatorial objects grow rapidly and reach to huge value. size.
 2. There is no algorithms available which can solve these problems in finite amount of time
 3. Many of these problems fall in the category of unsolvable problem.
- Some combinatorial problems can be solved by efficient algorithms.

6. Geometric Problems

- ✓ Geometric algorithms deal with geometric objects such as points ,lines and polygons.
- ✓ The procedure for solving a variety of geometric problems includes the problems of constructing simple geometric shapes such as triangles, circles and so on.

The two classic problems of computational geometry are the

2. Closest pair problem
3. Convex hull problem

- ✓ The closest pair problem is self explanatory. Given n points in the plane, find the closest pair among them.
- ✓ The convex hull problem is used to find the smallest convex polygon that would include all the points of a given set.
- ✓ The geometric problems are solved mainly in applications to computer graphics or in robotics

6.Numerical problems

- ✓ Numerical problems are problems that involve mathematical objects of continuous nature such as solving equations and systems of equations computing definite integrals evaluating functions and so on.
- ✓ Most of the mathematical problems can be solved approximate algorithms.
- ✓ These algorithms require manipulating of the real numbers; hence we may get wrong output many times.

3.Explain the fundamentals of the analysis framework. Or explain time-space trade off of the algorithm designed. April/May 2019

- Efficiency of an algorithm can be in terms of time or space.
- This systematic approach is modelled by a frame work called as analysis frame work.

Analysis framework

- The efficiency of an algorithm can be decided by measuring the performance of an algorithm.
- The performance of an algorithm is computed by two factors
 - amount of time required by an algorithm to execute
 - amount of storage required by an algorithm

Overview

- Space complexity
- Time complexity
- Measuring an Input's size
- Measuring Running Time
- Orders of Growth

Space complexity

- The space complexity can be defined as amount of memory required by an algorithm to run.

- To compute the space complexity we use two factors: constant and instance characteristics.
- The space requirement $S(p)$ can be given as $S(p) = C + S(p)$
Where C is a constant i.e. fixed part and it denotes the space of inputs and outputs.

Time complexity

- The time complexity of an algorithm is the amount of computer time required by an algorithm to run to completion.
- For instance in multiuser system, executing time depends on many factors such as
 - System load
 - Number of other programs running
 - Instruction set used
 - Speed underlying hardware
- The time complexity is therefore given in term of frequency count
 - Frequency count is a count denoting number of times of execution of statement

Example

```

For (i=0; i<n; i++)
{
    sum = sum + a[i];
}

```

Statement	Frequency count
i=0	1
i<n	This statement executes for (n+1) times. When conditions is true i.e. when i<n is true , the execution happens to be n times , and the statement execute once more when i<n is false
i++	n times
sum = sum + a[i]	n times
Total	3n + 2

Measuring an Input's size

- All algorithms run longer on larger inputs.
- Ex: Sorting larger arrays, multiply larger matrices etc.
- Investigates an algorithm efficiency as a function of some parameter n indicating the algorithm input size.
- Example:
 - In problem of evaluating a polynomial $p(x) = a_n x^n + \dots + a_0$ of degree n , the parameter will be the polynomial's degree or the number of its coefficients which is larger by one than its degree.
- In spell checking algorithm,
 - If algorithm examines the individual character of its input, then the size of the input is the no. of characters.
 - If the algorithm processes the word, the size of the input is the no. of words.

Measuring Running Time

- Some units of time measurement such as a second, a millisecond and so on can be used to measure the running time of a program implementing the algorithm.
- **Drawbacks**
 1. Dependence on the speed of a particular computer
 2. Dependence on the quality of a program implementing the algorithm.
 3. The compiler used in generating the machine code.

4. The difficulty of clocking the actual running time of the program
- Since we are in need to measure an algorithm's efficiency, we should have a metric that does not depend on these factors.
 - One possible approach is to count the number of times of the algorithm's operations is executed. But this approach is difficult and unnecessary.
 - The main objective is to identify the most important operation of the algorithm, called the **Basic Operation** - the operation contributing the most to the total running time, and compute the number of times the basic operation is executed.
 - It is not so difficult to identify the basic operation of an algorithm: it is usually the most time consuming operation in the algorithm's innermost loop.

Example

- Most sorting algorithms work by comparing the elements (keys) of a list being sorted with each other. For such algorithms the basic operation is a Key Comparison.

Problem statement	Input Size	Basic operation
Searching a key element from the list of n elements	List of n elements	Comparison of key with every element of list
Performing matrix multiplication	The two matrixes with order n×n	Actual multiplication of the elements in the matrices
Computing GCD of two numbers	Two numbers	Division

The formula to compute the execution time using basic operation is

$$T(n) \approx C_{op} C(n)$$

Where T(n) – running time

C(n) – no. of times this operation is executed.

Cop – execution time of algorithms basic operation.

Orders of Growth

- Measuring the performance of an algorithm in relation with the input size n is called order of growth.

Worst Case, Best Case and Average Case efficiencies

- It is reasonable to measure an algorithm's efficiency as a function of a parameter indicating the size of the algorithm's input.
- But for many algorithms the running time depends not only on an input size but also on the specifics of a particular input.

Example: Sequential Search or Linear Search AU: Dec -11, Marks 10

ALGORITHM *SequentialSearch(A[0..n - 1], K)*

//Searches for a given value in a given array by sequential search

//Input: An array A[0..n - 1] and a search key K

//Output: The index of the first element in A that matches K

// or -1 if there are no matching elements

i ← 0

while *i* < *n* **and** *A*[*i*] ≠ *K* **do**

i ← *i* + 1

if *i* < *n* **return** *i*

else return -1

- This algorithm searches for a given item using some search key K in a list of 'n' elements by checking successive elements of the list until a match with the search key

is found or the list is exhausted.

- The algorithm makes the largest number of key comparisons among all possible inputs of size n: $C_{\text{worst}}(n)=n$

Worst case efficiency

- The worst case efficiency of an algorithm is its efficiency for the worst case input of size n, which is an input (or inputs) of size n. For which the algorithm runs the longest among all possible of that size.
- The way to determine the worst case efficiency of an algorithm is that:
 - Analyse the algorithm to see what Kind of inputs yield the largest value of the basic operations count C(n) among all possible inputs of size n and then compute its value $C_{\text{worst}} = (n)$.

Best case efficiency

- The best case efficiency of an algorithm is its efficiency for the best case input of size n, which is an input (or inputs) of size n for which the algorithm runs the fastest among all possible inputs of that size.
- The way to determine the best case efficiency of an algorithm is as follows.
 - First, determine the kind of inputs of size n.
 - Then ascertain the value of C(n) on these inputs.
- **Example:** For sequential search, the best case inputs will be lists of size 'n' with their first elements equal to a search key: $C_{\text{best}}(n) = 1$.

Average case efficiency

- It yields the necessary information about an algorithm's behaviour on a "typical" or "random" input.
- To determine the algorithm's average case efficiency some assumptions about possible inputs of size 'n'.
- The average number of key comparisons $C_{\text{avg}}(n)$ can be computed as follows:
 - In case of a successful search the probability of the first match occurring in the position of the list is p/n for every i. and the number of comparisons made by the algorithm in such a situation is obviously 'i'.
 - In case of an unsuccessful search, the number of comparisons is 'n' with the probability of such a search being (1-p). Therefore,

$$C_{\text{avg}}(n) = \left[\frac{p}{n} \cdot 1 + \frac{p}{n} \cdot 2 + \dots + \frac{p}{n} \cdot i + \dots + \frac{p}{n} \cdot n \right] + n \cdot (1-p)$$

$$= \frac{p}{n} [1+2+3+\dots+i+\dots+n] + n(1-p)$$

$$= \frac{p \cdot n(n+1)}{2n} + n(1-p)$$

There may be n elements at which chances of 'not getting element' are possible. Hence n . (1-p)

$$C_{\text{avg}}(n) = \frac{p(n+1)}{2n} + n(1-p)$$

Example:

- If p = 1 (i.e.) if the search is successful, then the average number of key comparisons made by sequential search is (n+1)/2.
- If p = 0 (i.e.) if the search is unsuccessful, then the average number of key comparisons will be 'n' because the algorithm will inspect all n elements on all such inputs.

4. Explain the Asymptotic Notations and its properties? Or explain briefly Big oh notation , Omega notation and Theta notation give an example (Apr/May-2017) or what are the Rules of Manipulate Big-Oh Expression and about the typical growth rates of

algorithms? Nov/Dec 2017 Nov/Dec 2018

Define Big O notation, Big Omega and Big Theta Notation. Depict the same graphically and explain. May/June 2018 , Nov/Dec 2019

Explain the importance of asymptotic analysis for running time of an algorithm with an example. (April/May 2021)

Asymptotic notations are mathematical tools to represent time complexity of algorithms for measuring their efficiency. Types :

- Big Oh notation - 'O'
- Omega notation - 'Ω'
- Theta notation - 'Θ'
- Little Oh notation - 'o '

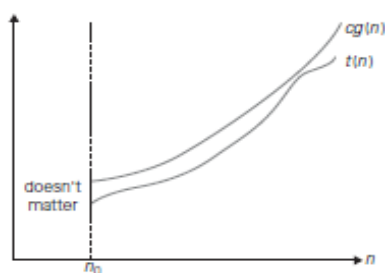
Big Oh notation (O)

- The big oh notation is denoted by 'O'.
- It is a method of representing the **upper bound of algorithm's running time**.
- Using big oh notation we can give **longest amount of time taken** by the algorithm to complete.

Definition

A function $t(n)$ is said to be in $O(g(n))$ ($t(n) \in O(g(n))$), if $t(n)$ is bounded above by constant multiple of $g(n)$ for all values of n , and if there exist a positive constant c and non negative integer n_0 such that

- $t(n) \leq c * g(n)$ for all $n \geq n_0$.



- 1 Big-oh notation: $t(n) \in O(g(n))$.

Example 1:

Consider function $t(n) = 2n + 2$ and $g(n) = n^2$. Then we have to find some constant c , so that $f(n) \leq c * g(n)$.

As $t(n) = 2n + 2$ and $g(n) = n^2$. Then we find c for $n=1$ then

$$\begin{aligned} t(n) &= 2n + 2 \\ &= 2(1) + 2 \end{aligned}$$

$$t(n) = 4$$

$$\begin{aligned} \text{And } g(n) &= n^2 \\ &= (1)^2 \end{aligned}$$

$$g(n) = 1$$

$$\text{i.e } t(n) > g(n)$$

if $n = 2$ then,

$$\begin{aligned} t(n) &= 2n + 2 \\ &= 2(2) + 2 \end{aligned}$$

$$t(n) = 6$$

$$\begin{aligned} \text{And } g(n) &= n^2 \\ &= (2)^2 \end{aligned}$$

$$g(n) = 4$$

$$\text{i.e } t(n) > g(n)$$

if $n = 3$ then,

$$t(n) = 2n + 2$$

$$\begin{aligned}
 &= 2(3) + 2 \\
 t(n) &= 8 \\
 \text{And } g(n) &= n^2 \\
 &= (3)^2 \\
 g(n) &= 9 \\
 \text{i.e. } t(n) &< g(n) \text{ is true.}
 \end{aligned}$$

Hence we can conclude that for $n > 2$, we obtain

$$t(n) < g(n)$$

Thus always upper bound of existing time is obtained by big oh notation.

Omega Notation (Ω)

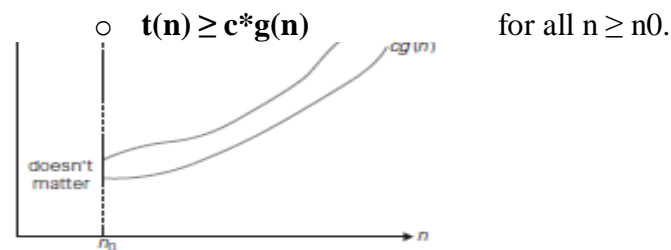
Omega notation is denoted by ' Ω '.

This notation is used to represent the **lower bound of algorithm's** running time.

Using omega notation we can denote **shortest amount of time taken** by algorithm.

Definition

A function $t(n)$ is said to be in $\Omega(g(n))$ ($t(n) \in \Omega(g(n))$), if $t(n)$ is bounded below by constant multiple of $g(n)$ for all values of n , and if there exist a positive constant c and non negative integer n_0 such that



■ Big-omega notation: $t(n) \in \Omega(g(n))$.

○ **Example 1:**

Consider $t(n) = 2n^2 + 5$ and $g(n) = 7n$

Then if $n = 0$
 $t(n) = 2(0)^2 + 5$
 $= 5$
 $g(n) = 7(0)$
 $= 0$ i.e. $t(n) > g(n)$

But if $n = 1$
 $t(n) = 2(1)^2 + 5$
 $= 7$
 $g(n) = 7(1)$
 $= 7$ i.e. $t(n) = g(n)$

But if $n = 2$
 $t(n) = 2(2)^2 + 5$
 $= 9$
 $g(n) = 7(2)$
 $= 12$ i.e. $t(n) < g(n)$

But if $n = 3$
 $t(n) = 2(3)^2 + 5$
 $= 18 + 5$
 $= 23$
 $g(n) = 7(3)$
 $= 21$ i.e. $t(n) > g(n)$

Thus for $n > 3$ we get $t(n) > c * g(n)$.

It can be represented as

$$2n^2 + 5 \in \Omega(n)$$

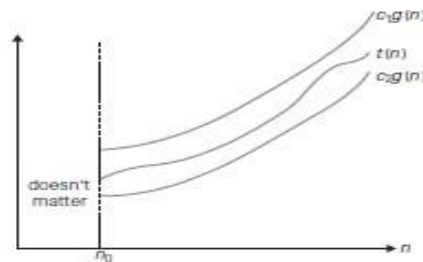
Theta Notation (Θ)

The theta notation is denoted by Θ . By this method the **running time is between upper bound and lower bound**.

Definition

A function $t(n)$ is said to be in $\Theta(g(n))$ ($t(n) \in \Theta(g(n))$), if $t(n)$ is bounded both above and below by constant multiple of $g(n)$ for all values of n , and if there exist a positive constant c_1 and c_2 and non negative integer n_0 such that

$$c_2 * g(n) \leq t(n) \leq c_1 * g(n) \quad \text{for all } n \geq n_0.$$



- 3 Big-theta notation: $t(n) \in \Theta(g(n))$.

Example 1:

If $t(n) = 2n + 8$ and $g(n) = 7n, 5n$

Where $n \geq 2$

$$c_2 * g(n) \leq t(n) \leq c_1 * g(n) \quad \text{for all } n \geq$$

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

$$(t(n) \in \Theta(g(n)))$$

Similarly $t(n) = 2n + 8$

$$g(n) = 7n$$

$$g(n) = 5n$$

$$\text{i.e. } 5n < 2n + 8 < 7n \quad \text{for } n \geq 2$$

Here $c_2 = 5$ and $c_1 = 7$ with $n_0 = 2$

Little oh notation (o)

The function $t(n) = o(g(n))$, if $O(g(n))$ and $t(n) \ll \Omega(g(n))$

Example

$$t(n) = 3n+2$$

$$\text{Where } n > 0, 3n+2 \leq 5n^2$$

By definition of Big Oh

$$t(n) = Cg(n)$$

$$C = 5; g(n) = n^2$$

$$\text{But } t(n) = 3n+2 \ll \Omega(n^2)$$

$$\text{Therefore } t(n) = 3n+2 = o(n^2)$$

Useful property involving the Asymptotic notation:

The following property is useful in analyzing algorithms that comprise two consecutively executed parts.

Theorem

If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$ then,

$$t_1(n) + t_2(n) \in (\max \{g_1(n), g_2(n)\})$$

Proof

Since $t_1(n) \in O(g_1(n))$, there exist some constant C_1 and some non negative integer n_1 such that

$$t_1(n) \leq C_1 (g_1(n)) \text{ for all } n \geq n_1$$

Since

$$t_2(n) \in O(g_2(n))$$

$$t_2(n) \leq C_2 (g_2(n)) \text{ for all } n \geq n_2$$

Let us denote,

$$C_3 = \max \{C_1, C_2\} \text{ and}$$

Consider $n \geq \max \{n_1, n_2\}$, so that both the inequalities can be used.

The addition of two inequalities becomes,

$$\begin{aligned} t_1(n) + t_2(n) &\leq C_1 (g_1(n)) + C_2 (g_2(n)) \\ &\leq C_3 (g_1(n)) + C_3 (g_2(n)) \\ &\leq C_3 2 \max \{g_1(n), (g_2(n))\} \end{aligned}$$

Hence,

$$t_1(n) + t_2(n) \in O(\max \{g_1(n), g_2(n)\}),$$

with the constants C and n_0 required by the definition being $2C_3 = 2 \max (C_1, C_2)$ and $\max \{n_1, n_2\}$ respectively.

The property implies that the algorithms overall efficiency will be determined by the part with a larger order of growth.

(i.e.) its least efficient part is

$$\begin{aligned} t_1(n) \in O(g_1(n)) & \quad t_1(n) + t_2(n) \in O(\max \{g_1(n), g_2(n)\}) \\ t_2(n) \in O(g_2(n)) & \end{aligned}$$



Using limits for comparing orders of growth

There are 3 principal cases,

	0,	Implies that (n) has a smaller order of growth than g(n)	
$\lim_{n \rightarrow \infty} t(n)/g(n)$	C,	Implies that (n) has a same order of growth than g(n)	}
	∞ ,	Implies that (n) has a larger order of growth than g(n)	

L' Hospital's rule.

$$\lim_{n \rightarrow \infty} t(n)/g(n) = \lim_{n \rightarrow \infty} t'(n)/g'(n)$$

Stirling's formula

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \text{ for large values of } n.$$

Asymptotic Growth Rate

Three notations used to compare orders of growth of an algorithm's basic operation count

- $O(g(n))$: class of functions $f(n)$ that grow no faster than $g(n)$
- $\Omega(g(n))$: class of functions $f(n)$ that grow at least as fast as $g(n)$
- $\Theta(g(n))$: class of functions $f(n)$ that grow at same rate as $g(n)$

Example

1. Compare orders of growth of $\frac{1}{2}n(n-1)$ and n^2

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\frac{1}{2}n(n-1)}{n^2} &= \frac{1}{2} \lim_{n \rightarrow \infty} \frac{n^2 - n}{n} \\ &= \frac{1}{2} \lim_{n \rightarrow \infty} \left(1 - \frac{1}{n}\right) \\ &= \frac{1}{2} \end{aligned}$$

Since the limit is equal to a positive constant, the functions have the same order of growth or symbolically

$$\frac{1}{2}n(n-1) \in \theta(n^2)$$

2. Compare orders of growth of $\log_2 n$ and \sqrt{n}

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{\log_2 n}{\sqrt{n}} &= \lim_{n \rightarrow \infty} \frac{(\log_2 n)}{(\sqrt{n})} \\ &= \lim_{n \rightarrow \infty} \frac{(\log_2 e)^{\frac{1}{n}}}{\frac{1}{2\sqrt{2}}} \\ &= 2 \log_2 e \lim_{n \rightarrow \infty} \frac{\sqrt{n}}{n} \\ &= 0 \end{aligned}$$

Since the limit is equal to zero, $\log_2 e$ has a smaller order of growth than \sqrt{n} or symbolically,

$$\log_2 e \in o(\sqrt{n}).$$

Little Oh notation is rarely used in analysis of algorithms.

3. Compare orders of growth $n!$ and 2^n .

By using the Stirling's formula,

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{n!}{2^n} &= \lim_{n \rightarrow \infty} \frac{\sqrt{2\pi n} \left(\frac{e}{n}\right)^n}{2^n} \\ &= \lim_{n \rightarrow \infty} \sqrt{2\pi n} \frac{n^n}{2^n e^n} \\ &= \lim_{n \rightarrow \infty} \sqrt{2\pi n} \left(\frac{n}{2e}\right)^n \\ &= \infty \end{aligned}$$

Basic Asymptotic Efficiency Classes

Class	Name	Comments
1	Constant	Short of best-case efficiencies
\log_n	Logarithmic	Cutting a problem size by a constant factor
n	Linear	Algorithms that scan a list of size n . (eg sequential search)
$n \log_n$	$n \log n$	Many divide and conquer algorithm
n^2	Quadratic	Efficiency of algorithm with two embedded loops.
n^3	Cubic	Efficiency of algorithm with three embedded loops.
2^n	Exponential	Generate all the subsets of an n element set.

n!	Factorial	Algorithm that generate all permutations of an n element set
----	-----------	--

5. Explain the Mathematical analysis for non-recursive algorithm or write an algorithm for determining the uniqueness of an array. Determine the time complexity of your algorithm. (Apr/May-2017) April/May 2019

General plan for analyzing efficiency of non-recursive algorithm

1. Decide the **input size** based on parameter n.
2. Identify the algorithm **basic operation(s)**.
3. Check whether the number of times the **basic operation is executed** depends on only on the size of the input.
4. Set up a **sum** expressing the number of times the algorithm basic operation is excited
5. Simplify the sum using standard formula and rules

Example 1: Problem for finding the value of the largest element in a list of n numbers

The pseudo code to solving the problem is

```

ALGORITHM MaxElement(A[0..n-1])
//Problem Description : This algorithm is for finding the
//maximum value element from the array
//Input:An array A[0..n-1] of real numbers
//Output: Returns the largest element from array
Maxval ← A[0]
For i ← 1 to n-1 do
{
    If ( A[i]>max_value)then
        Maxval ← A[i]
}
Return Max_value

```

Searching the maximum element from an array

If any value is large than **current Max_Value** then set new **Max_value** by obtained larger value

Mathematical Analysis

Step 1: The input size is the number of elements in the array(ie.),n

Step 2 : The basic operation is comparison in loop for finding larger value There are two operations in the for loop

- ✓ Comparison operation a[i]->maxval
- ✓ Assigment operation maxval->a[i]

Step 3: The comparison is executed on each repetition of the loop. As the comparison is made for each value of n there is no need to find best case worst case and average case analysis.

Step 4: Let C(n) be the number of times the comparison is executed.

The algorithm makes comparison each time the loop executes.

That means with each new value of I the comparison is made.

Hence for i= 1 to n – 1 times the comparison is made . therefore we can formulate C(n) as

$$C(n) = \text{one comparison made for each value of } i$$

Step 5 : let us simplify the sum

$$\begin{aligned} \text{Thus } C_{(n)} &= \sum_{i=1}^{n-1} 1 \\ &= n-1 \in \theta(n) \end{aligned}$$

Using the rule $\sum_{i=1}^n 1 = n \in \theta(n)$

The frequently used two basic rules of sum manipulation are,

$$\sum_{i=1}^u C a_i = C \sum_{i=1}^u a_i \quad \text{R1}$$

$$\sum_{i=1}^u (a_i + b_i) = \sum_{i=1}^u a_i + \sum_{i=1}^u b_i \quad \text{R2}$$

The two summation formulas are

$$1. \quad \sum_{i=1}^n 1 = u - l + 1$$

Where $l \leq u$ are some lower and upper integer limits S1

$$2. \quad \begin{aligned} \sum_{i=0}^n i &= \sum_{i=1}^n i = 1 + 2 + \dots + n \\ &= n(n+1)/2 \\ &= 1/2 n^2 \in \theta(n^2) \end{aligned} \quad \text{S2}$$

Example 2: Element uniqueness problem-check whether all the element in the list are distinct April/May 2019

ALGORITHM UniqueElements(A[0..n-1])

//Checks whether all the elements in a given array are distinct

//Input :An array A[0..n-1]

//Output Returns 'true' if all elements in A are distinct and 'false'

//otherwise

for i ← 0 to n-2 do

for j ← i+1 to n-1 do

if a[i] = a[j] then

return false

else

return true

If any two elements in the array are similar then return .false indicating that the array elements

Mathematical analysis

Step 1: Input size is n i.e total number of elements in the array A

Step 2: The basic iteration will be comparison of two elements . this operation the innermost operation in the loop . Hence

if a[i] = a[j] then comparison will be the basic operation .

Step 3 : The number of comparisons made will depend upon the input n .

but the algorithm will have worst case complexity if the same element is located at the end of the list. Hence the basic operation depends upon the input n and worst case

Worst case investigation

Step 4: The worst case input is an array for which the number of elements comparison $C_{\text{worst}}(n)$ is the largest among the size of the array.

There are two kinds of worst case inputs, They are

1. Arrays with no equal elements.

2. Arrays in which the last two elements are pair of equal elements.

For the above inputs, one comparison is made for each repetition of the inner most loop (ie) for each value of the loop's variable 'j' between its limits i+1 and n-1 and this is repeated limit for each values of the outer loop (ie) for each value of the loop's variable 'i' between 0 and n-2. Accordingly,

$$C_{\text{worst}}(n) = \text{Outer loop} \times \text{Inner loop}$$

$$C_{\text{worst}}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

Step 5: now we will simplify C_{worst} as follows

$$= \sum_{i=0}^{n-1} [(n-1) - (i+1) + 1] \quad \left\{ \sum_{i=k}^n [1 = n - k + 1 \text{ is the rule}] \right.$$

$$= \sum_{i=0}^{n-2} [(n-1-i)]$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i$$

Now taking (n-1) as a common factor, we can write

$$= (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2}$$

This can be obtained using formula $\sum_{i=1}^n i = n(n+1)/2$

$$= (n-1) \sum_{i=0}^{n-2} \left(1 - \frac{(n-2)(n-1)}{2} \right)$$

$$= (n-1)^2 - \frac{(n-2)(n-1)}{2}$$

This can be obtained using formula $\sum_{i=1}^n 1 = (n-1+1) = n$ i.e when $\sum_{i=0}^{n-1} 1 = (n-2) - 0 + 1 = (n-1)$

Solving this equation we will get

$$= 2(n-1)(n-1) - (n-2)(n-1)/2$$

$$= (2(n^2 - 2n + 1) - (n^2 - 3n + 2)) / 2$$

$$= ((n^2 - n) / 2)$$

$$= 1/2 n^2$$

$$\in \Theta(n^2)$$

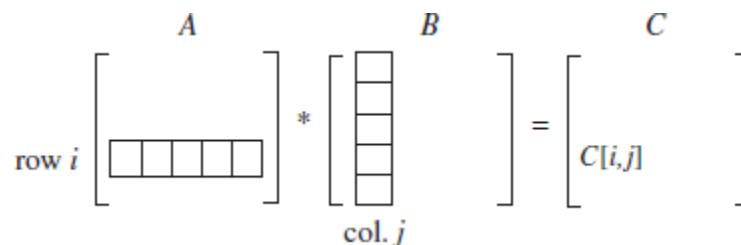
We can say that in the worst case the algorithm needs to compare all $n(n-1)/2$ distinct of its n elements.

Therefore $C_{\text{worst}}(n) = 1/2n^2 \in \Theta(n^2)$

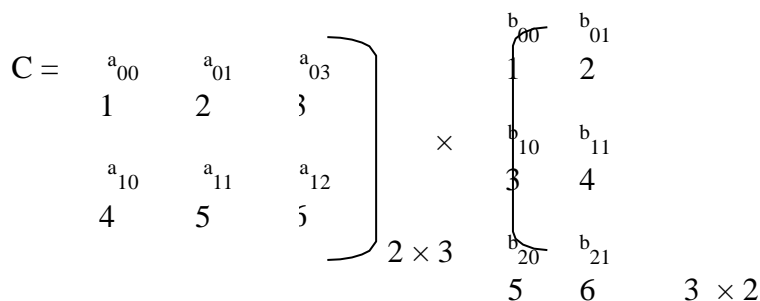
EXAMPLE 3 : Obtaining matrix multiplication

Given two $n \times n$ matrices A and B, find the time efficiency of the definition-based algorithm for computing their product $C = AB$, where A and B are n by n ($n \times n$) matrices.

By definition, C is an $n \times n$ matrix whose elements are computed as the scalar (dot) products of the rows of matrix A and the columns of matrix B:



where $C[i, j] = A[i, 0]B[0, j] + \dots + A[i, k]B[k, j] + \dots + A[i, n-1]B[n-1, j]$ for every pair of indices $0 \leq i, j \leq n-1$.



The formula for multiplication of the above two matrices is

$$a_{00} \times b_{00} + a_{01} \times b_{10} + a_{02} \times b_{20} \quad a_{00} \times b_{01} + a_{01} \times b_{11} + a_{02} \times b_{21}$$

$$C = \begin{matrix} a_{10} \times b_{00} + a_{11} \times b_{10} + a_{12} \times b_{20} & a_{10} \times b_{01} + a_{11} \times b_{11} + a_{12} \times b_{21} \\ \\ \\ \end{matrix} \left[\begin{matrix} \\ 1 \times 2 + 2 \times 4 + 3 \times 6 \\ 4 \times 2 + 5 \times 4 + 6 \times 6 \end{matrix} \right]$$

$$C = \begin{matrix} 1 \times 1 + 2 \times 3 + 3 \times 5 \\ 4 \times 1 + 5 \times 3 + 6 \times 5 \end{matrix}$$

$$C = \begin{bmatrix} 22 & 28 \\ 49 & 64 \end{bmatrix}$$

Now the algorithm for matrix multiplication is

```

ALGORITHM MatrixMultiplication(A[0..n - 1, 0..n - 1], B[0..n - 1, 0..n - 1])
//Multiplies two square matrices of order n by the definition-based algorithm
//Input: Two n x n matrices A and B
//Output: Matrix C = AB
for i ← 0 to n - 1 do
    for j ← 0 to n - 1 do
        C[i, j] ← 0.0
        for k ← 0 to n - 1 do
            C[i, j] ← C[i, j] + A[i, k] * B[k, j]
return C
    
```

Mathematical analysis

Step 1: The input's size of above algorithm is simply order of matrices i.e n.

Step 2: The basic operation is in the innermost loop and which is

$$C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$$

There are two arithmetical operations in the innermost loop here

1. Multiplication
2. Addition

Step 3: The basic operation depends only upon input size. There are no best case, worst case and average case efficiencies. Hence now we will go for computing **sum**. There is just one multiplication which is repeated no each execution of innermost loop. (**a for loop using variable k**). Hence we will compute the efficiency for innermost loops.

Step 4: The sum can be denoted by M (n).

$$M(n) = \text{outermost} \times \text{inner loop} \times \text{innermost loop} \text{ (1 execution)}$$

$$= [\text{for loop using i}] \times [\text{for loop using j}] \times [\text{for loop using k}] \times (1 \text{ execution})$$

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1.$$

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n$$

$\sum_{j=1}^{n-1} = n$

 \leftarrow

$$= \sum_{i=0}^{n-1} n^2$$

$$M(n) = n^3$$

Thus the simplified sum is n^3 . Thus the time complexity of matrix multiplication $\Theta(n^3)$

Running time of the Algorithm T(n)

The estimation of running time of the algorithm on a particular machine is calculated by using the product.

$$T(n) \approx c_m M(n) = c_m n^3$$

Where- c_m is the time of one multiplication on the machine in question.

We would get a more accurate estimate if we took into account the time spent on the additions, too:

$$T(n) \approx c_m M(n) + c_a A(n) = c_m n^3 + c_a n^3 = (c_m + c_a) n^3$$

$$T(n) \approx c_m M(n) = c_m n^3$$

where c_m is the time of one multiplication on the machine in question. We would get a more accurate estimate if we took into account the time spent on the additions, too:

Time spend addition CA (n)

The time speed to perform the addition operation is given by

$$T(n) = c_a A(n) = c_a n^3$$

Where

c_a is the time taken to perform the one addition.

Hence the running time of the algorithm is given by

$$T(n) \approx c_m M(n) + c_a A(n) = c_m n^3 + c_a n^3 = (c_m + c_a) n^3$$

The estimation differs only by the multiplication constants and not by the order of growth.

EXAMPLE 4: The following algorithm finds the number of binary digits in the binary representation of a positive decimal integer.

ALGORITHM *Binary(n)*

//Input: A positive decimal integer n

//Output: The number of binary digits in n's binary representation

count ← 1

while *n > 1 do*

count ← count + 1

n ← [n/2]

return *count*

Mathematical analysis

Step 1: The input size is n i.e. . The positive integer whose binary digit in binary representation needs to be checked.

Step 2 : The basic operation is denoted by while loop. And it is each time checking whether $n > 1$. The while loop will be executed for the number of time at which $n > 1$ is true . it will be executed once more when $n > 1$ is false . but when $n > 1$ is false the statements inside while loop wont get executed.

Step 3: The value of n is halved on each repetition of the loop. Hence efficiency algorithm is equal to $\log_2 n$

Step 4: hence total number of times the while loop gets executed is $[\log_2 n] + 1$

Hence time complexity for counting number of bits of given number is $\Theta(\log_2 n)$. this indicates floor value of $\log_2 n$

6. Explain the Mathematical analysis for recursive algorithm. (Apr/May-2017) or

Discuss the steps in Mathematical analysis for recursive algorithms. Do the same for finding factorial of a number. Nov/Dec 2017 or solve the following recurrence equations using iterative method or tree Nov/Dec 2019

Discuss various methods used for mathematical analysis of recursive algorithms. May/June 2018**General plan for analyzing efficiency of recursive algorithms**

1. Decide the input size based on parameter n .
2. Identify algorithms basic operations
3. Check how many times the basic operation is executed.
To find whether the **execution of basic operation** depends upon the input size n . **determine worst, average, and best case** for input of size n . if the basic operation depends upon worst case average case and best case then that has to be analyzed separately.
4. Set up the **recurrence relation** with some initial condition and expressing the basic operation.
5. Solve the recurrence or at least determine the order of growth. While solving the recurrence we will use the **forward and backward substitution method**. And then correctness of formula can be proved with the help of **mathematical induction** method.

Example 1: Computing factorial of some number n .

To compare the factorial $F(n)=n!$ for an arbitrary non negative integer

$$\begin{aligned}
 N! &= 1.2.3.....(n-1).n \\
 &= (n-1)! \cdot n, \text{ for } n \geq 1 \\
 0! &= 1
 \end{aligned}$$

By definition $F(n)=F(n-1)! \cdot n$

ALGORITHM $F(n)$

```

//Computes  $n!$  recursively
//Input: A nonnegative integer  $n$ 
//Output: The value of  $n!$ 
if  $n = 0$  return 1
else return  $F(n - 1) * n$ 

```

Mathematical Analysis:

Step 1: The algorithm's input size is n .

Step 2: The algorithm's basic operation in computing factorial is multiplication.

Step 3 : The recursive function call can be formulated as

According to the formula, $F(n)$ is computed as

$$F(n) = F(n-1) * n, \quad \text{for } n > 0$$

And the number of execution is denoted by $M(n)$.

The number of multiplication $M(n)$ is computed as

$$M(n) = M(n-1) + 1, \quad \text{for } n > 0$$

To compute $F(n-1)$

To multiply $F(n-1)$ by n

$M(n-1)$ multiplication are spent to compute $F(n-1)$.

One more multiplication is needed to multiply the result by n .

Step 4: in step 3 the recurrence relation is obtained.

The equation is

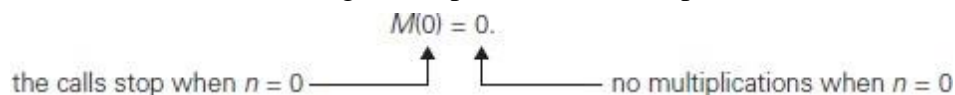
$$M(n)=M(n-1) +1, \quad \text{for } n > 0$$

Defines $M(n)$ not explicitly (i.e.) as a function of n , but implicitly as function of its value at another point, namely $n-1$. These equations are called as **recurrence relations or recurrences**.

- Recurrences relations play an important role in the analysis of algorithm and some area of applied mathematics.
- To solve a recurrence relation $M(n)=M(n-1)+1$ the formula for the sequence $M(n)$ in terms of n only should be find.
- To determine the unique solution, an initial condition is needed that tells the value with which the sequence starts.
- The initial value is obtained from the condition if $n=0$ return 1 that makes the algorithm stops.

The condition, if $n=0$ return 1 tells 2 things

1. The recursive call stops when $n=0$ the smallest value for which the algorithm is executed. Hence $M(n)=0$.
2. When $n=0$ the algorithm performs no multiplication



Forward Substitution:

$$M(1) = M(0) + 1$$

$$M(2) = M(1) + 1 = 1 + 1 = 2$$

$$M(3) = M(2) + 1 = 2 + 1 = 3$$

The recurrence relation and the initial condition for the algorithm number of multiplication $M(n)$ is

$$M(n) = M(n-1) + 1, \text{ for } n \geq 1, M(0) = 0$$

Backward substitution:

$$M(n) = M(n-1) + 1$$

$$\text{Substitute } M(n-1) = M(n-2) + 1$$

Now $M(n)$ becomes

$$M(n) = [M(n-2) + 1] + 1$$

$$= M(n-2) + 2$$

$$\text{Substitute } M(n-2) = M(n-3) + 1$$

Now $M(n)$ becomes

$$M(n) = [M(n-3) + 1] + 2$$

$$= M(n-3) + 3$$

From the substitution method we can establish a general formula as :

$$M(n) = M(n-i) + i;$$

Since $n=0$, substitute $i=n$;

Now let us prove correctness of this formula using mathematical induction as follows

Proof

$M(n) = n$ by using mathematical induction

Basis : let $n = 0$ then

$$M(n) = 0$$

$$\text{i.e } M(0) = 0 = n$$

Induction: if we assume $M(n - 1) = n - 1$ then

$$M(n) = M(n - 1) + 1$$

$$= n - 1 + 1$$

$$= n$$

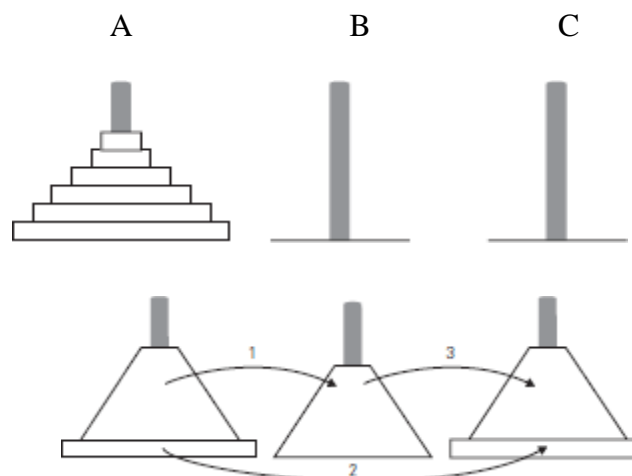
i.e $M(n) = n$ Thus the time complexity of factorial function is $\Theta(n)$

Give the general plan for Analyzing the time efficiency of Recursive Algorithms and use recurrence to find number of moves for Towers of Hanoi problem. May/June 2018

Example 2: Tower of Hanoi puzzle

- ✓ In this puzzle, there are n disks of different sizes, and three pegs.
- ✓ Initially all the disks are on the first peg in order of size, the largest on the bottom and the smallest on the top.
- ✓ The goal is to move all the disks from peg 1 to peg 3 using peg 2 as auxiliary.
- ✓ One disk should be moved at a time and do not place a larger disk on top of a smaller one.
- ✓ The following steps are used to move $n > 1$ disks from peg 1 to peg 3, peg 2 as auxiliary.
 1. Move $n - 1$ disks recursively from peg 1 to peg 3. (peg 2 as auxiliary).
 2. Move the largest disk directly from peg 1 to peg 3.
 3. Move $n - 1$ disks recursively from peg 2 to peg 3. (peg 2 as auxiliary).

For example, if $n = 1$ then the single disk is moved from source peg to destination peg directly.



General plan to tower of Hanoi problem

The input size is the number of disks “ n ”.

The algorithm basic operation is moving one disk at a time.

The number of moves $M(n)$ depends only on n .

The recurrence equation is,

$$M(n) = M(n-1) + 1 + M(n-1), \text{ for } n > 1;$$

$$M(n) = 2M(n-1) + 1, \text{ for } n > 1;$$

The initial condition $M(1) = 1$

Now the recurrence relation for number of moves is,

$$M(n) = 2M(n-1) + 1, \text{ for } n > 1$$

$$M(1) = 1$$

The recurrence relation is solved by using backward substitution method

Backward substitution Method

$$M(n) = 2M(n-1) + 1$$

Substitute

$$M(n-1) = 2M(n-2) + 1$$

$$M(n) = 2[2M(n-2) + 1] + 1$$

$$M(n) = 2^2M(n-2) + 2 + 1$$

Substitute

$$M(n-2) = 2M(n-3) + 1$$

Now, $M(n)$ becomes

$$M(n) = 2^2[2M(n-3) + 1] + 2 + 1$$

$$M(n) = 2^3[M(n-3) + 2^2 + 2 + 1]$$

Hence after i substitution $M(n)$ becomes

$$M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + 2^{i-3} + \dots + 2 + 1$$

$$= 2^i M(n-i) + 2^i - 1$$

Therefore the general formula is $2^i M(n-i) + 2^i - 1$

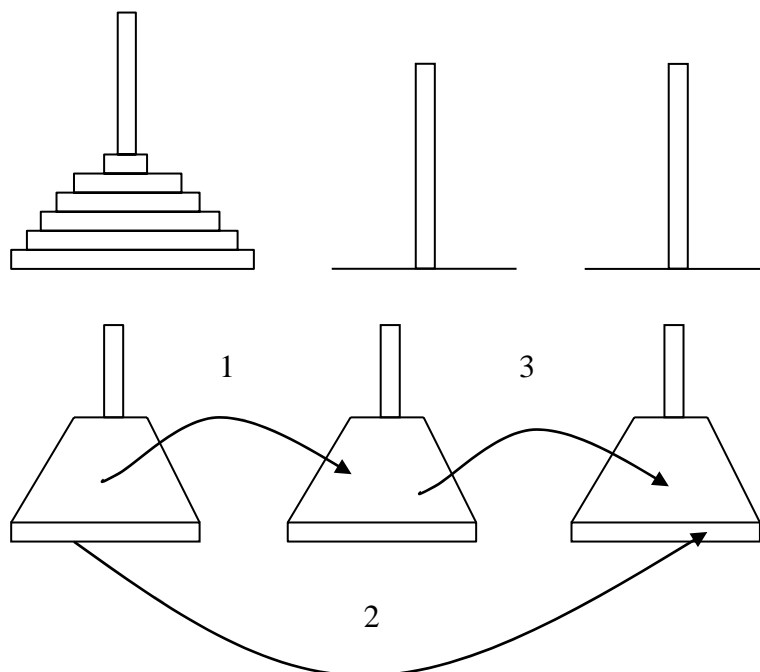


Fig. recursive solution to the Tower of Hanoi puzzle

Solution to recurrence relation is

Since the initial condition is $n=1$ becomes $i=n-1$.

The recurrence relation is

$$M(n)=2^i M(n-i)+2^i-1 \dots \dots \dots (1)$$

Substitute $i=n-1$ in (1)

$$\begin{aligned} M(n) &= 2^{n-1} M(n-(n-1)) + 2^{n-1} - 1 \\ &= 2^{n-1} M(1) + 2^{n-1} - 1 \\ &= 2^{n-1} + 2^{n-1} - 1 \\ &= 2^n - 1 \end{aligned}$$

$M(n) = 2^n - 1$ Thus this is an exponential algorithm, It runs unimaginably long time for moderate values of n .

Example 3 :To find the number of binary digits in binary representation

Algorithm BinRec(n)

//**Input:** A positive decimal integer n

//**Output:** The number of binary digits in n 's binary representation

if $n=1$

 return 1

else

 return BinRec($\lfloor n/2 \rfloor$)+1

Recurrence and Initial Condition

A Recurrence for the number of addition $A(n)$ made by the algorithm is the number of addition made in computing BinRec($\lfloor n/2 \rfloor$) is $A(\lfloor n/2 \rfloor)$ plus one more addition is made

Thus recurrence is

$$A(n) = A(\lfloor n/2 \rfloor) + 1, \text{ for } n \geq 1$$

$A(n)$ -> number of addition made by the algorithm

$A(\lfloor n/2 \rfloor)$ -> number of addition made to compute $A(\lfloor n/2 \rfloor)$

The recursive call end when n is equal to 1 and no addition is made.

The initial condition is $A(1) = 0$

To solve the recurrence, backward substitutions cannot be used. The reason is the presence on $\lfloor n/2 \rfloor$ in the functions argument and the value of n is not power of 2.

A theorem called **Smoothness rule** is used to solve the recurrence.

The standard approach for solving such recurrence is to solve it only for $n = 2^k$.

The order of growth observed for $n = 2^k$ gives a correct answer about the order of growth of all values of n .

$n = 2^k$ takes the form

$$A(2^k) = A(2^{k-1}) + 1 \text{ for } k > 0,$$

$$A(2^0) = 0.$$

Now, backward substitutions can be applied.

Backward Substitution Method

$$A(2^k) = A(2^{k-1}) + 1$$

$$\begin{aligned} \text{substitute } A(2^{k-1}) &= A(2^{k-2}) + 1 \\ &= [A(2^{k-2}) + 1] + 1 \\ &= A(2^{k-2}) + 2 \end{aligned}$$

$$\begin{aligned} \text{substitute } A(2^{k-2}) &= A(2^{k-3}) + 1 \\ &= [A(2^{k-3}) + 1] + 2 \\ &= A(2^{k-3}) + 3 \dots \dots \end{aligned}$$

After i iteration

$$\begin{aligned} A(2^k) &= A(2^{k-i}) + i \\ &= A(2^{k-k}) + k \\ &= A(2^0) + k \\ &= A(1) + k \end{aligned}$$

Thus, we end up with

$$A(2^k) = A(1) + k = k$$

After returning to the original variable

$$n = 2^k \text{ and hence } k = \log_2 n,$$

$$A(n) = \log_2 n \in \Theta(\log n)$$

Example 4: Fibonacci series

A sequence of Fibonacci numbers is 0,1,1,2,3,5,8,13,21,34.....

The Fibonacci sequence can be defined by the simple recurrence

$$F(n) = F(n-1) + F(n-2), \text{ for } n > 1 \dots \dots \dots 1$$

The two initial conditions are

$$F(0) = 0$$

$$F(1) = 1$$

Explicit formula for the n^{th} Fibonacci number

Backward substitution method is not used to solve the recurrence $F(n) = F(n-1) + F(n-2)$, for $n > 1$, because which fails to produce easily discernible pattern.

So, the theorem that describes solution to a homogeneous second order linear recurrence with constant coefficient is used to solve the problem.

The homogenous with constant coefficient is

$$ax(n)+bx(n-1)+cx(n-2)=0..... (2)$$

Where,

a,b,c are fixed real numbers called the coefficients of recurrence and a≠0

x(n) is the unknown sequence to be found

The characteristics equation of the recurrence equation is

$$Ar^2+br+c=0 (3)$$

The recurrence relation can be written as

$$F(n)-F(n-1)-F(n-2)=0..... (4)$$

The characteristics equation for (4)

$$r^2-r-1=0$$

The roots are

$$R_{1,2} = \frac{-1 \pm \sqrt{1 - 4(1)}}{2}$$

$$R_{1,2} = \frac{1 \pm \sqrt{5}}{2}$$

$$R_1 = \frac{1 + \sqrt{5}}{2}$$

$$R_2 = \frac{1 - \sqrt{5}}{2}$$

The characteristics equation has two distinct real roots.

Now the recurrence relation is

$$X(n)=\alpha r_1^n+\beta r_2^n..... (5)$$

Substitute r1 and r2 in (5),

$$F(n)=\alpha\left(\frac{1 + \sqrt{5}}{2}\right)^n+\beta\left(\frac{1 - \sqrt{5}}{2}\right)^n.....(6)$$

Now substitute the value of f(0) and F(1) in equation(6)

$$F(0) = \alpha\left(\frac{1 + \sqrt{5}}{2}\right)^0+\beta\left(\frac{1 - \sqrt{5}}{2}\right)^0=0(7)$$

$$F(1) = \alpha\left(\frac{1 + \sqrt{5}}{2}\right)^1+\beta\left(\frac{1 - \sqrt{5}}{2}\right)^1=0(8)$$

By solving equation (7) and (8),the linear equation in two unknown α and β

$$\alpha + \beta = 0$$

$$\alpha\left(\frac{1 + \sqrt{5}}{2}\right) + \beta\left(\frac{1 - \sqrt{5}}{2}\right) = 0(11)$$

(11)-(10) gives

$$\left(\frac{1+\sqrt{5}}{2}\right)\beta - \left(\frac{1+\sqrt{5}}{2}\right)\beta = -1$$

$$\frac{\beta}{2} + \frac{\sqrt{5}}{2}\beta - \frac{\beta}{2} + \frac{\sqrt{5}}{2}\beta = -1$$

$$2 \frac{\sqrt{5}}{2}\beta = -1$$

$$\beta = -\frac{\sqrt{5}}{2}$$

Substitute $\beta = -\frac{\sqrt{5}}{2}$ in (9)

$$\alpha + \beta = 0$$

$$\alpha - \frac{1}{\sqrt{5}} = 0$$

$$\alpha = \frac{1}{\sqrt{5}} \quad \beta = -\frac{\sqrt{5}}{2}$$

Substitute the value of α and β in equation (6)

$$F(n) = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2}\right)^n \right]$$

$$F(n) = \frac{1}{\sqrt{5}} [\phi^n - \phi^{-n}]$$

Where

$$\Phi = \frac{1+\sqrt{5}}{2}$$

$$\Phi = 1.61803$$

$$\Phi^{-1} = \frac{1}{\Phi}$$

$$\Phi^{-1} = 0.61803$$

The constant Φ is known as, Golden Ratio.

The value of Φ^{-1} lies between -1 and 0.

When n goes to infinity, Φ^{-1} gets infinitely small value. So, it can be omitted.

Therefore $F(n) = \frac{1}{\sqrt{5}} \Phi^n$

So, for every non negative n , $F(n) = \frac{1}{\sqrt{5}} \Phi^n$ is rounded to the nearest integer.

Algorithm for computing Fibonacci numbers

First method

Algorithm F(n)

//Computes the nth Fibonacci number recursively by using its definition.

//Input: A nonnegative integer n

//Output: The nth Fibonacci number

if $n < 1$

return n

Else

return F(n-1)+(n-2)

the algorithm's basic operation is addition.

Let A(n) is the number of additions performed by the algorithm to compute F(n).

The number of additions needed to compute F(n-1) is A(n-1) and the number of additions needed to compute F(n-2) is A(n-2).

The algorithm needs one more addition to compute the sum of A(n-1) and A(n-2).

Thus the recurrence for A(n) is

$$A(n)=A(n-1) + A(n-2)+1, \text{ for } n>1$$

$$A(0)=0$$

$$A(1)=0$$

The recurrence $A(n)-A(n-1)-A(n-2)=1$ is same as $F(n)-F(n-1)-F(n-2)=0$, but its right hand side not equal to zero. These recurrences are called **inhomogeneous recurrences**.

General techniques are used to solve inhomogeneous recurrences.

The inhomogeneous recurrences is converted into homogeneous recurrence by rewriting the in homogeneous recurrence as, $A(n)+1]-[A(n-1)+1]-[A(n-2)+1]=0$ (14)

Now substitute, $B(n)=A(n)+1$

Now (14) becomes, $B(n)-B(n-1)-B(n-2)=0$

$$B(0)=0$$

$$B(1)=1$$

Here $B(n)=F(n+1)$

Since $B(n)=A(n)+1$

$$B(n-1)=A(n)$$

So $A(n)=B(n)-1$

Substitute $F(n+1)-1$ (15)

We know that

$$F(n)=\frac{1}{\sqrt{5}}(\phi^n - \phi^{-n})$$

$$F(n+1)=\frac{1}{\sqrt{5}}(\phi^{n+1} - \phi^{-n-1}) \dots\dots\dots(16)$$

Substitute (16) in (15)

$$A(n)=\frac{1}{\sqrt{5}}(\phi^{n+1} - \phi^{-n-1}) - 1$$

Hence

$$A(n) \in \Theta(\phi^n)$$

The poor efficiency class of algorithm could be anticipated from the class of recurrence

The reason behind the algorithm inefficiency can be traced by looking at the tree of recursive calls $n=6$

The same values of the function are evaluated again and again which is extremely inefficiently.

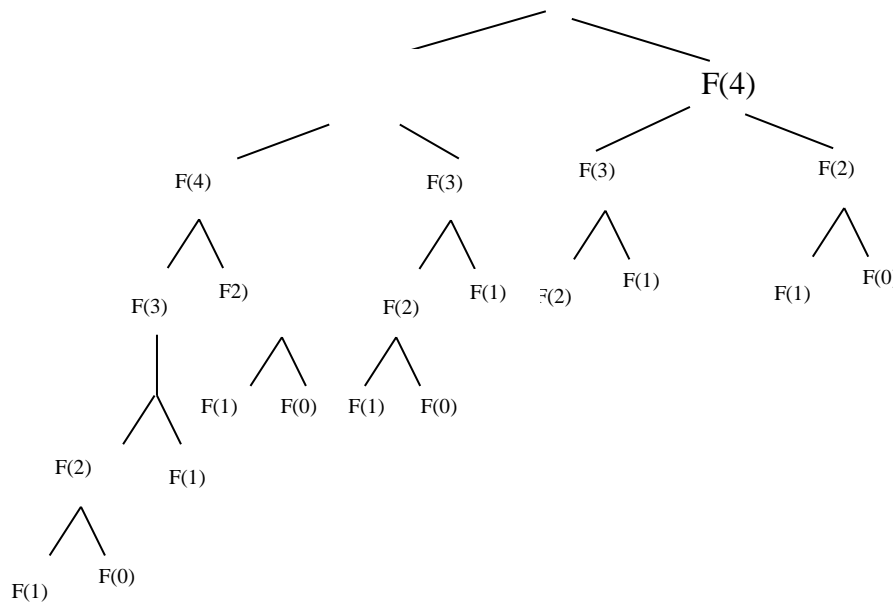


Fig Tree of recursive calls for computing the Fibonacci number for $n = 6$

7. Find the time complexity and space complexity of the following problems. Factorial using recursion and compute the nth Fibonacci number using iterative statements. Dec 2012

8. Solve the following recurrence relations: or solve the following recurrence equation:

$$T(n) = T(n/2) + 1, \text{ where } n = 2^k \text{ for all } k \geq 0$$

$$T(n) = T(n/3) + T(2n/3) + cn, \text{ where 'c' is a constant and 'n' is the input size.}$$

Dec 2012 April/May 2019

$$1. T(n) = \begin{cases} 2T(n/2) + 3 & n > 2 \\ 2 & n = 2 \end{cases}$$

$$\begin{aligned} T(n) &= 2T(n/2) + 3 \\ &= 2\{2T(n/4) + 3\} + 3 \\ &= 4T(n/4) + 6 \\ &= 4\{2T(n/8) + 3\} + 6 \\ &= 8T(n/8) + 9 \end{aligned}$$

 $= 2^k T(n/2^k) + 3n$
 $T(n) = n \log n + 3n$
 Time complexity = $O(n \log n)$

2. $T(n) = \begin{cases} 2T(n/2) + cn & n > 1 \\ a & n = 1 \end{cases}$ where a and c constants

$T(n) = 2T(n/2) + cn$
 $= 2\{(2T(n/2) + cn)/2\} + cn$
 $= 2\{(2T(n/4) + cn/2)\} + cn$

 $= 4T(n/4) + cn + cn$
 $= 4\{(2T(n/8) + cn/4)\} + cn + cn$

 $= 8T(n/8) + cn + cn + cn$

 $= 2^k T(n/2^k) + k(cn)$
 $T(n) = n \log n + k(cn)$
 Time complexity = $O(n \log n)$

8. Show the following equalities are correct June 2013

- i. $5n^2 - 6n = \Theta(n^2)$
 - ii. $n! = O(n^n)$
 - iii. $n^3 + 10^6 n^2 = \Theta(n^3)$
 - iv. $2n^2 2^n + n \log n = \Theta(n^2 2^n)$
 - i. $5n^2 - 6n = \Theta(n^2) \Rightarrow$ highest order of growth is n^2
 - ii. $n! = O(n^n) \Rightarrow$ highest order of growth $O(n)$
 - iii. $n^3 + 10^6 n^2 = \Theta(n^3) \Rightarrow$ highest order of growth $O(n^3)$
 - iv. $2n^2 2^n + n \log n = \Theta(n^2 2^n) \Rightarrow$ highest order of growth $O(n^2)$
- Nov 2010

9. Prove that for any two functions $f(n)$ and $g(n)$, we have $f(n) \rightarrow \Theta(g(n))$ if and only if $f(n) \rightarrow O(g(n))$ and $f(n) \rightarrow \Omega(g(n))$ Nov 2010

Given function:
 $f(n)$ and $g(n)$
 $f(n) = O(g(n))$ when $f(n) \leq C_1 g(n)$ for all $n \geq n_0$ (1)
 $f(n) = \Omega(g(n))$ when $f(n) \geq C_2 g(n)$ for all $n \geq n_0$ (2)
 from (1) and (2)
 $C_2 g(n) \leq f(n) \leq C_1 g(n)$ for all $n \geq n_0$ (3)
 (i.e) $\Theta(g(n)) = O(g(n)) \Omega(g(n))$
From (3) $f(n) = \Theta(g(n))$ hence proved

10. (a) If you have to solve the searching problem for a list of n numbers, how can you take

advantage of the fact that the list is known to be sorted? Give separate answers for lists represented as arrays lists represented as linked lists. (AU april/may 2015)

For a sorted array do a binary search to divide the array in half for each query, thus $O(\lg n)$. If the list is linked you must you do a linear search which is $O(n)$, unless you use a linked binary search tree, which is $O(\lg n)$

11. The best-case analysis is not as important as the worst-case analysis of an algorithm". Yes or No ? Justify your answer with the help of an example. (April/May 2021)

The Best Case analysis is bogus. **Guaranteeing a lower bound on an algorithm doesn't provide any information as** in the worst case, an algorithm may take years to run. For some algorithms, all the cases are asymptotically the same, i.e., there are no worst and best cases. For example, Merge Sort.

11. Derive the worst case analysis of merge sort using suitable illustration (AU april/may 2015)
Efficiency of Merge Sort

- In merge sort algorithm the two recursive calls are made. Each recursive call focuses on $n/2$ elements of the list .
- After two recursive calls one call is made to combine two sublist i.e to merge all n elements.
- Hence we can write recurrence relation as

$$T(n) = T(n/2) + T(n/2) + cn$$

$T(n/2)$ = Time taken by left sublist

$T(n/2)$ = time taken by right sublist

$T(n)$ = time taken for combining two sublists

where $n > 1$ $T(1) = 0$

The time complexity of merge sort can be calculated using two methods

- Master theorem
- Substitution method

Master theorem Let , the recurrence relation for merge sort is

$$T(n) = T(n/2) + T(n/2) + cn$$

Let $T(n) = aT(n/b) + f(n)$ be a recurrence relation

$$\text{i.e. } T(n) = 2T(n/2) + cn \text{ -----(1)}$$

$$T(1) = 0 \text{ -----(2)}$$

As per master theorem $T(n) = \Theta(n^d \log n)$ if $a = b$

As equation (1), $a = 2$, $b = 2$ and $f(n) = cn$ and $a = b^d$ i.e $2 = 2^1$

This case gives us , $T(n) = \Theta(n \log_2 n)$

Hence the average and worst case time complexity of merge sort is

$$C_{\text{worst}}(n) = (n \log_2 n)$$

Substitution method Let, the recurrence relation for merge sort be

$$T(n) = T(n/2) + T(n/2) + cn \quad \text{for } n > 1$$

$$\text{i.e. } T(n) = 2T(n/2) + cn \quad \text{for } n > 1 \text{ -----(3)}$$

$$T(1) = 0 \text{ -----(4)}$$

Let us apply substitution on equation (3) .

Assume $n=2^k$

$$T(n) = 2T(n/2) + cn$$

$$T(n) = 2T(2^k/2) + c.2^k$$

$$T(2^k) = 2T(2^{k-1}) + c.2^k$$

If $k = k-1$ then,

$$T(2^k) = 2T(2^{k-1}) + c.2^k$$

$$T(2^k) = 2[2T(2^{k-2}) + c.2^{k-1}] + c.2^k$$

$$T(2^k) = 2^2 T(2^{k-2}) + 2.c.2^{k-1} + c.2^k$$

$$T(2^k) = 2^2 T(2^{k-2}) + 2.c.2^k /2 + c.2^k$$

$$T(2^k) = 2^2 T(2^{k-2}) + c.2^k + c.2^k$$

$$T(2^k) = 2^2 T(2^{k-2}) + 2c .2^k$$

Similarly we can write,

$$T(2^k) = 2^3 T(2^{k-3}) + 3c .2^k$$

$$T(2^k) = 2^4 T(2^{k-4}) + 4c .2^k$$

.....

....

$$T(2^k) = 2^k T(2^{k-k}) + k.c.2^k$$

$$T(2^k) = 2^k T(2^0) + k.c.2^k$$

$$T(2^k) = 2^k T(1) + k.c.2^k \text{ -----(5)}$$

But as per equation (4), $T(1) = 0$

There equation (5) becomes ,

$$T(2^k) = 2^k .0 + k . c . 2^k$$

$$T(2^k) = k . c . 2^k$$

But we assumed $n=2^k$, taking logarithm on both sides.i.e. $\log_2 n = k$

$$\text{Therefore } T(n) = \log_2 n . cn$$

$$\text{Therefore } T(n) = \Theta(n \log_2 n)$$

Hence the average and worst case time complexity of merge sort is

$$C_{\text{worst}}(n) = (n \log_2 n)$$

Time complexity of merge sort

Best case	Average case	Worst case
$\Theta(n \log_2 n)$	$\Theta(n \log_2 n)$	$\Theta(n \log_2 n)$

12.write Insertion sort algorithm and estimate its running time.

- ✓ Like selection sort, insertion sort loops over the indices of the array. It just calls `insert` on the elements at indices $1, 2, 3, \dots, n-1$. Just as each call to `indexOfMinimum` took an amount of time that depended on the size of the sorted subarray, so does each call to `insert`. Actually, the word "does" in the previous sentence should be "can," and we'll see why.
- ✓ Let's take a situation where we call `insert` and the value being inserted into a subarray is less than every element in the subarray.
- ✓ For example, if we're inserting 0 into the subarray $[2, 3, 5, 7, 11]$, then every element in the subarray has to slide over one position to the right. So, in general, if we're inserting into a subarray with k elements, all k might have to slide over by one position.
- ✓ Rather than counting exactly how many lines of code we need to test an element against a key and slide the element, let's agree that it's a constant number of lines; let's call that constant c . Therefore, it could take up to $c \cdot k$ lines to insert into a subarray of k elements.
- ✓ Suppose that upon every call to `insert`, the value being inserted is less than every element in the subarray to its left. When we call `insert` the first time, $k=1$. The second time, $k=2$. The third time, $k=3$. And so on, up through the last time, when $k=n-1$.

Therefore, the total time spent inserting into sorted subarrays

$$\text{is } c \cdot 1 + c \cdot 2 + c \cdot 3 + \dots + c \cdot (n-1) = c \cdot (1 + 2 + 3 + \dots + (n-1))$$

That sum is an arithmetic series, except that it goes up to $n-1$ rather than n . Using our formula for arithmetic series, we get that the total time spent inserting into sorted subarrays is

$$c \cdot (n-1+1)((n-1)/2) = cn^2/2 - cn/2.$$

Using big- Θ notation, we discard the low-order term $cn/2$ and the constant factors c and $1/2$, getting the result that the running time of insertion sort, in this case, is $\Theta(n^2)$.

Can insertion sort take *less* than $\Theta(n^2)$ time? The answer is yes. Suppose we have the array $[2, 3, 5, 7, 11]$, where the sorted subarray is the first four elements, and we're inserting the value 11. Upon the first test, we find that 11 is greater than 7, and so no elements in the subarray need to slide over to the right.

- ✓ Then this call of `insert` takes just constant time. Suppose that *every* call of `insert` takes constant time. Because there are $n-1$ calls to `insert`, if each call takes time that is some constant c , then the total time for insertion sort is $c \cdot (n-1)$ which is $\Theta(n)$, not $\Theta(n^2)$.
- ✓ Can either of these situations occur? Can each call to `insert` cause every element in the subarray to slide one position to the right? Can each call to `insert` cause no elements to slide? The answer is yes to both questions.
- ✓ A call to `insert` causes every element to slide over if the key being inserted is less than every element to its left. So, if every element is less than every element to its left, the running time of insertion sort is $\Theta(n^2)$.
- ✓ What would it mean for every element to be less than the element to its left? The array would have to start out in *reverse* sorted order, such as $[11, 7, 5, 3, 2]$. So a reverse-sorted array is the worst case for insertion sort.
- ✓ How about the opposite case? A call to `insert` causes no elements to slide over if the key being inserted is greater than or equal to every element to its left. So, if every element is greater than or equal to every element to its left, the running time of insertion sort is $\Theta(n)$.
- ✓ This situation occurs if the array starts out already sorted, and so an already-sorted array is the best case for insertion sort.

What else can we say about the running time of insertion sort? Suppose that the array starts out in a random order. Then, on average, we'd expect that each element is less than half the elements to its left.

- ✓ In this case, on average, a call to `insert` on a subarray of k elements would slide $k/2$ of them. The running time would be half of the worst-case running time. But in asymptotic notation, where constant coefficients don't matter, the running time in the average case would still be $\Theta(n^2)$, just like the worst case.
- ✓ What if you knew that the array was "almost sorted": every element starts out at most some constant number of positions, say 17, from where it's supposed to be when sorted?
- ✓ Then each call to `insert` slides at most 17 elements, and the time for one call of `insert` on a subarray of k elements would be at most $17 \cdot k$. Over all $n-1$ calls to `insert`, the running time would be $17 \cdot (n-1)$, which is $\Theta(n)$, just like the best case. So insertion sort is fast when given an almost-sorted array.

To sum up the running times for insertion sort:

- **Worst case:** $\Theta(n^2)$.
- **Best case:** $\Theta(n)$.
- **Average case for a random array:** $\Theta(n^2)$.
- **"Almost sorted" case:** $\Theta(n)$.

If you had to make a blanket statement that applies to all cases of insertion sort, you would have to say that it runs in $O(n^2)$ time. You cannot say that it runs in $\Theta(n^2)$ time in all cases, since the best case runs in $\Theta(n)$ time. And you cannot say that it runs in $\Theta(n)$ time in all cases, since the worst-case running time is $\Theta(n^2)$.

13. Show how to implement a stack using two queues. Analyze the running time of the stack operations.

Show how to implement a stack using two queues. Analyze the running time of the stack operations.

- The pseudocode is as follows.

```
public class TwoQueueStack
{
    Queue q1;
    Queue q2;
    int flag = 0;
    // 0: the stack is empty;
    // 1: all data are stored in q1;
    // 2: all data are stores in q2;

    // always push into the empty queue, then move all data from
    // the other queue to the current queue.
    // always pop element from the queue containing data.
    // suppose q1 and q2 are implemented by linked list. The queues never
    // get full.

    public void push(Object e)
    {
        switch(flag)
        {
            case 0: q1.enqueue(e);
                    flag = 1;
                    break;
            case 1: q2.enqueue(e);
                    while (!q1.isEmpty())
                        q2.enqueue(q1.dequeue());
                    flag = 2;
                    break;
            case 2: q1.enqueue(e);
                    while (!q2.isEmpty())
                        q1.enqueue(q2.dequeue());
                    flag = 1;
                    break;
            default: error 'illegal state';
        }
    }

    public Object pop()
    {
        switch(flag)
        {
            case 0: error 'underflow -- stack is empty, can't pop';
            case 1: retElement = q1.dequeue();
                    if (q1.isEmpty())
                        flag = 0;
                    return retElement;
            case 2: retElement = q2.dequeue();
                    if (q2.isEmpty())

                                flag = 0;
                                return retElement;
            default: error 'illegal state';
        }
    }
}
```

- The running time for push operation is $O(n)$. The running time for pop operation is $\Theta(1)$.

14. find the closest asymptotic tight bound by solving the recurrence equation

$T(n)=8T(n/2)+n^2$ with $(T(1)=1)$ using recursion tree method.[Assume that $T(1)\in\Theta(1)$]

Example: Solve $T(n) = 8T(n/2) + n^2$ ($T(1) = 1$)

$$\begin{aligned}
 T(n) &= n^2 + 8T(n/2) \\
 &= n^2 + 8(8T(\frac{n}{2^2}) + (\frac{n}{2})^2) \\
 &= n^2 + 8^2T(\frac{n}{2^2}) + 8(\frac{n^2}{4}) \\
 &= n^2 + 2n^2 + 8^2T(\frac{n}{2^2}) \\
 &= n^2 + 2n^2 + 8^2(8T(\frac{n}{2^3}) + (\frac{n}{2^2})^2) \\
 &= n^2 + 2n^2 + 8^3T(\frac{n}{2^3}) + 8^2(\frac{n^2}{4^2}) \\
 &= n^2 + 2n^2 + 2^2n^2 + 8^3T(\frac{n}{2^3}) \\
 &= \dots \\
 &= n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \dots
 \end{aligned}$$

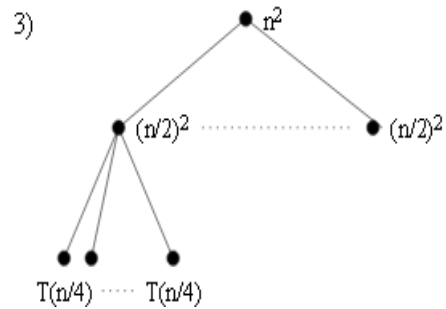
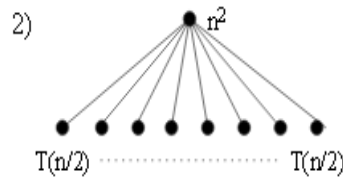
- Recursion depth: How long (how many iterations) it takes until the subproblem has constant size? i times where $\frac{n}{2^i} = 1 \Rightarrow i = \log n$
- What is the last term? $8^i T(1) = 8^{\log n}$

$$\begin{aligned}
 T(n) &= n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \dots + 2^{\log n - 1}n^2 + 8^{\log n} \\
 &= \sum_{k=0}^{\log n - 1} 2^k n^2 + 8^{\log n} \\
 &= n^2 \sum_{k=0}^{\log n - 1} 2^k + (2^3)^{\log n}
 \end{aligned}$$

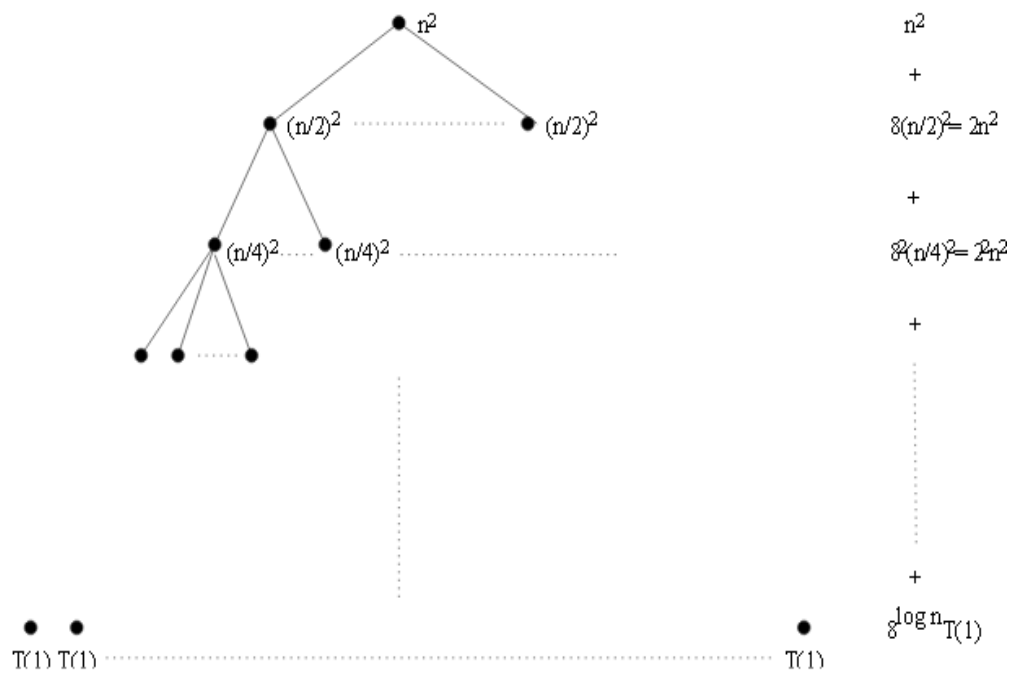
- Now $\sum_{k=0}^{\log n - 1} 2^k$ is a geometric sum so we have $\sum_{k=0}^{\log n - 1} 2^k = \Theta(2^{\log n - 1}) = \Theta(n)$
- $(2^3)^{\log n} = (2^{\log n})^3 = n^3$

$$\begin{aligned}
 T(n) &= n^2 \cdot \Theta(n) + n^3 \\
 &= \Theta(n^3)
 \end{aligned}$$

- we draw out the recursion tree with cost of single call in each node—running time is sum of costs in all nodes
- if you are careful drawing the recursion tree and summing up the costs, the recursion tree is a direct proof for the solution of the recurrence, just like iteration and substitution
- Example: $T(n) = 8T(n/2) + n^2$ ($T(1) = 1$)



$\log n$)



$$T(n) = n^2 + 2n^2 + 2^2n^2 + 2^3n^2 + 2^4n^2 + \dots + 2^{\log n - 1}n^2 + 8^{\log n}$$

15. Derive a loose bound on the following equation: $F(x) = 35x^8 - 22x^7 + 14x^5 - 2x^4 - 4x^2 + x - 15$

$$f(x) = 35x^8 - 22x^7 + 14x^5 - 2x^4 - 4x^2 + x - 15$$

Solution : Let, $f(n)$ and $g(n)$ are two non-negative functions.

Let c be some constant.

The equation

$$f(n) \leq c * g(n)$$

then $f(n) \in O(g(n))$ with tight bound.

But if $f(n) < c * g(n)$

then $f(n) \in O(g(n))$ with loose bound.

Consider the function

$$f(x) = 35x^8 - 22x^7 + 14x^5 - 2x^4 - 4x^2 + x - 15$$

and $g(x) = x^8$

If $x = 1$, then

$$f(x) = 35(1)^8 - 22(1)^7 + 14(1)^5 - 2(1)^4 - 4(1)^2 + 1 - 15$$

$$f(x) = 7$$

$$g(x) = x^8 = (1)^8$$

If we assume $c = 35$ then

We will always get

$$f(x) < g(x) \text{ for } x \geq 1$$

16. Solve the recurrence relations

$$X(n) = x(n-1) + 5 \text{ for } n > 1 \quad x(1) = 0$$

$$X(n) = 3x(n-1) \text{ for } n > 1 \quad x(1) = 4$$

$$X(n) = x(n-1) + n \text{ for } n > 0 \quad x(0) = 0$$

$$X(n) = x(n/2) + n \text{ for } n > 1 \quad x(1) = 1 \text{ (solve for } n = 2^k)$$

$$X(n) = x(n/3) + 1 \text{ for } n > 1 \quad x(1) = 1 \text{ (solve for } n = 3^k)$$

$$\mathbf{X(n) = x(n-1) + 5 \text{ for } n > 1 \quad x(1) = 0}$$

$$\mathbf{X(1) = 0}$$

$$\mathbf{\text{If } n=2}$$

$$\mathbf{X(2) = x(2-1) + 5}$$

$$= x(1) + 5$$

$$= 0 + 5$$

$$= 5$$

$$\mathbf{\text{If } n=3}$$

$$\mathbf{X(3) = x(3-1) + 5}$$

$$\begin{aligned}
 &=x(2)+5 \\
 &=5+5 \\
 &=10
 \end{aligned}$$

If $n=4$

$$\begin{aligned}
 X(4) &=x(4-1)+5 \\
 &=x(3)+5 \\
 &=10+5 \\
 &=15.....
 \end{aligned}$$

17. Use the most appropriate notation to indicate the time efficiency class of sequential search algorithm in the worst case, best case and the average case.

Solution : Sequential search

“Given a target value and a random list of values, find the location of the target in the list, if it occurs, by checking each value in the list in turn”

```

get (NameList, PhoneList, Name)
i = 1
N = length(NameList)
Found = FALSE
while ( (not Found) and (i <= N) ) {
  if ( Name == NameList[i] ) {
    print (Name, "s phone number is ", PhoneList[i])
    Found = TRUE
  }
  i = i+1
}
if ( not Found ) { print (Name, "s phone number not found!") }

```

Central unit of work: operations that occur most frequently

Central unit of work in sequential search:

Comparison of target Name to each name in the list

Also add 1 to i

Typical iteration: two steps (one comparison, one addition)

Given a large input list:

Best case: smallest amount of work algorithm must do

Worst case: greatest amount of work algorithm must do

Average case: depends on likelihood of different scenarios occurring

- **Best case:** target found with the first comparison (**1 iteration**)
- **Worst case:** target never found or last value (N iterations)
- **Average case:** if each value is equally likely to be searched, work done varies from 1 to N , on average $N/2$ iterations

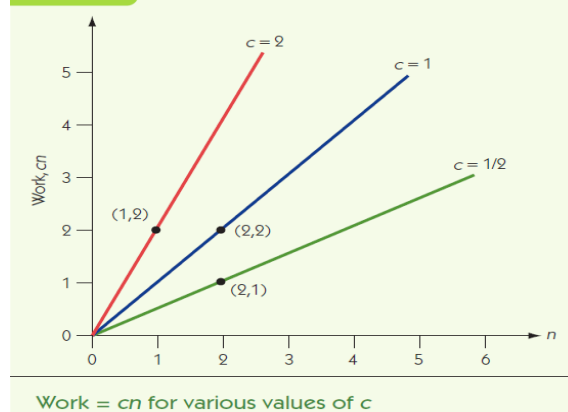
Sequential search worst case (N) grows linearly in the size of the problem $2N$ steps (one comparison and one addition per loop) Also some initialization steps...

On the last iteration, we may print something...After the loop, we test and maybe print...

To simplify analysis, disregard the “negligible” steps (which don’t happen as often), and ignore the coefficient in $2N$ Just pay attention to the dominant term (N)

Order of magnitude $O(N)$: the class of all linear functions (any algorithm that takes $C_1N + C_2$ steps for any constants C_1 and C_2)

FIGURE 3.4



18.(i) Prove that if $g(n)$ is $\Omega(f(n))$ then $f(n)$ is $O(g(n))$. May/June 2018

$f(n) \in \Omega(g(n)) \Leftrightarrow g(n) \in O(f(n))$

Proof:

$$O(f(n)) = \{g: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c, n_0 \in \mathbb{N} \forall n \geq n_0: g(n) \leq c \cdot f(n)\}$$

$$\Omega(g(n)) = \{f: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c, n_0 \in \mathbb{N} \forall n \geq n_0: f(n) \geq c \cdot g(n)\}$$

Step 1/2: $f(n) \in \Omega(g(n)) \Leftrightarrow g(n) \in O(f(n))$

$$\exists c, n_0 \in \mathbb{N} \forall n \geq n_0: f(n) \geq c \cdot g(n) \Rightarrow f(n)g(n) \geq c \Rightarrow 1g(n) \geq cf(n) \Rightarrow g(n) \leq 1c \cdot f(n)$$

And this is exactly the definition of $O(f(n))$.

Step 2/2: $f(n) \in \Omega(g(n)) \Leftarrow g(n) \in O(f(n))$

$$\exists c, n_0 \in \mathbb{N} \forall n \geq n_0: g(n) \leq c \cdot f(n) \Rightarrow \dots \Rightarrow f(n) \geq 1c \cdot g(n)$$

Hence proved.

19. Explain briefly about Empirical Analysis of Algorithm.

The principal alternative to the mathematical analysis of an algorithm's efficiency is its empirical analysis. This approach implies steps spelled out in the following plan.

General Plan for the Empirical Analysis of Algorithm Time Efficiency

1. Understand the experiment's purpose.
2. Decide on the efficiency metric M to be measured and the measurement unit (an operation count vs. a time unit).
3. Decide on characteristics of the input sample (its range, size, and so on).
4. Prepare a program implementing the algorithm for the experimentation.
5. Generate a sample of inputs.
6. Run the algorithm (or algorithms) on the sample's inputs and record the data observed.
7. Analyze the data obtained.

1. Purpose:

- To ensure theoretical assertion about the algorithm's efficiency
- comparing the efficiency of several algorithms for solving the same problem or different implementations of the same algorithm
- developing a hypothesis about the algorithm's efficiency class
- ascertaining the efficiency of the program implementing the algorithm on a particular machine.

2. how & What to measure

- Include a variable counter, to count the number of times the algorithm's basic operation is executed.
- In the implementing the algorithm, measure the running time of basic operation

Example

- In unix, the system command time may be used.
- computing the difference between the two($t_{\text{finish}} - t_{\text{start}}$).

Disadvantages of Measuring the system time

1. System's time is typically not very accurate, and you might get somewhat different results on repeated runs of the same program on the same inputs. An obvious remedy is to make several such measurements and then take their average (or the median) as the sample's observation point.
2. In the high speed of modern computers, the running time may fail to register at all and be reported as zero. The standard trick to overcome this obstacle is to run the program in an extra loop many times, measure the total running time, and then divide it by the number of the loop's repetitions.
3. The computer running under a time-sharing system such as UNIX, the reported time may include the time spent by the CPU on other programs, which obviously defeats the purpose of the experiment. Therefore, you should take care to ask the system for the time devoted specifically to execution of your program. (In UNIX, this time is called the "user time," and it is automatically provided by the time command.)

Advantage of Measuring physical running time

- (i) the physical running time provides very specific information about an algorithm's performance in a particular computing environment
 - (ii) Measuring time spent on different segments of a program can pinpoint a bottleneck in the program's performance that can be missed by an abstract deliberation about the algorithm's basic operation profiling.
4. Deciding on a sample of inputs

Sample size: (it is sensible to start with a relatively small sample and increase it later if necessary)

Range of input sizes: (typically neither trivially small nor excessively large)

- procedure for generating instances in the range chosen.
 - The instance sizes can either adhere to some pattern (e.g., 1000, 2000, 3000, . . . , 10,000 or 500, 1000, 2000, 4000, . . . , 128,000) or be generated randomly within the range chosen.
 - Several instances of the same size should be included or not.
5. Generate a sample of inputs (random numbers)

Typically, its output will be a value of a (pseudo)random variable uniformly distributed in the interval between 0 and 1. If a different (pseudo)random variable is desired, an appropriate transformation needs to be made. For example, if x is a continuous random variable uniformly distributed on the interval $0 \leq x < 1$, the variable $y = l + [x(r-l)]$ will be uniformly distributed among the integer values between integers l and $r-1$ ($l < r$).

Alternatively, you can implement one of several known algorithms for generating (pseudo)random numbers. The most widely used and thoroughly studied of such algorithms is the linear congruential method

ALGORITHM

Random(n, m, seed, a, b)

//Generates a sequence of n pseudorandom numbers according to the linear

// c o n g r u e n t i a l m e t h o d

//Input: A positive integer n and positive integer parameters m, seed, a, b

//Output: A sequence r_1, \dots, r_n of n pseudorandom integers uniformly

// distributed among integer values between 0 and $m-1$

//Note: Pseudorandom numbers between 0 and 1 can be obtained

// by treating the integers generated as digits after the decimal point

$r_0 \leftarrow \text{seed}$

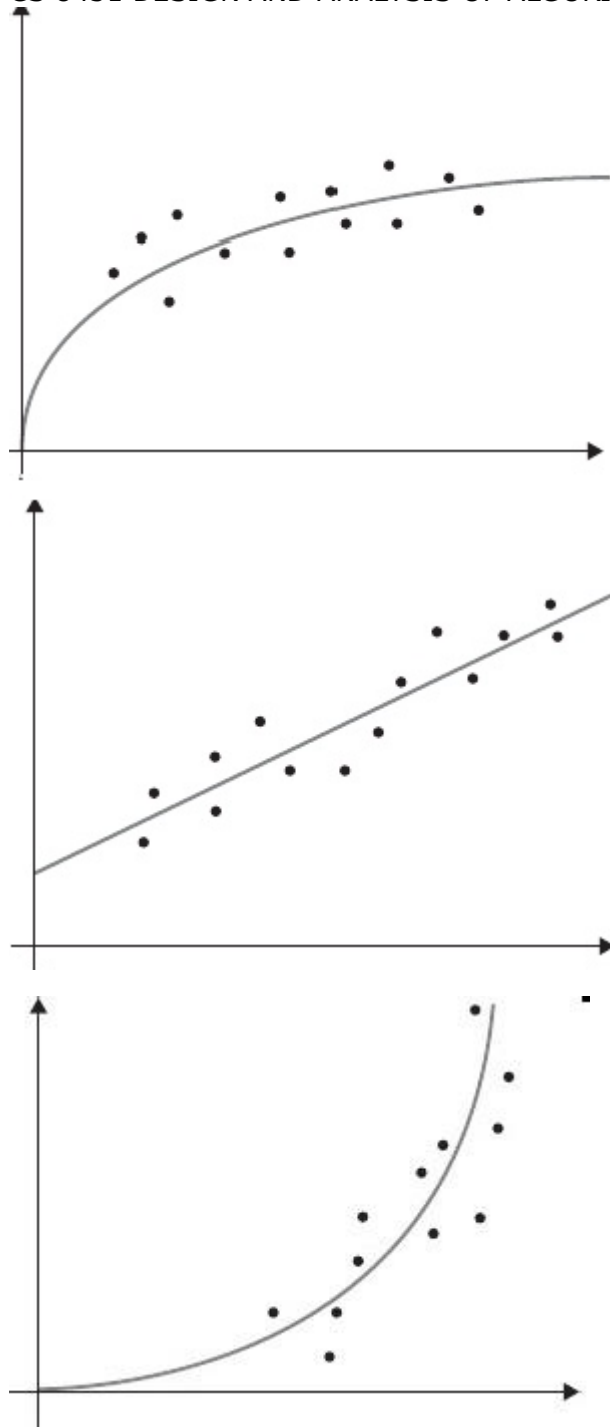
for $i \leftarrow 1$ **to** n **do**

$r_i \leftarrow (a * r_{i-1} + b) \bmod m$

6. Data analysis

- It is a good idea to use both these options whenever it is feasible because both methods have their unique strengths and weaknesses.
- The advantages of tabulated data lies in the opportunity to manipulate it easily and to find efficiency class of the algorithm.
- The Scatter plot representation helps in the analysis of algorithm efficiency class as given in figure

Shape of the scatter plot	Efficiency class
Concave shape	Logarithmic
Point around straight line or between two straight line	Linear
Convex shape	Quadratic and $n \log n$
Convex shape with rapid increase in the metrics valus	Cubic



Typical scatter plots. (a) Logarithmic. (b) Linear. (c) One of the convex functions

Application:

1. Predicting the algorithm performance on a sample size not included in the experiment sample.
2. The standard techniques of statistical data analysis and prediction can also be done.

20. Explain briefly about Algorithm Visualization.

Algorithm visualization is defined as the use of images to convey some useful information about algorithms. That information can be a visual illustration with the following combinations.

1. Algorithm's operation on different kinds of inputs
2. Same input for different algorithms to compare the execution speed.

An algorithm visualization uses graphic elements—points, line segments, two- or three-dimensional bars, and so on—to represent some “interesting events” in the algorithm's operation.

There are two principal variations of algorithm visualization:

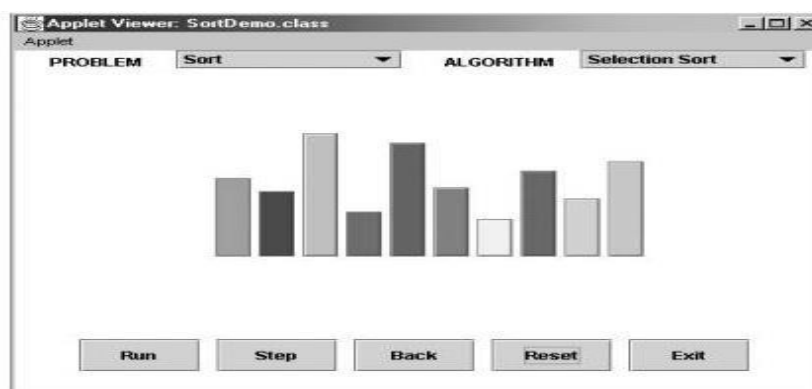
1. Static algorithm visualization
2. Dynamic algorithm visualization, also called algorithm animation

Static algorithm visualization shows an algorithm's progress through a series of still images. Algorithm animation, on the other hand, shows a continuous, movie-like presentation of an algorithm's operations. Animation is an arguably more sophisticated option, which, of course, is much more difficult to implement.

The features of an animations user interface was suggested by Peter Gloor is listed below

- Be consistent
- Be Interactive
- Be clear and concise
- Be forgiving to the user
- Adapt to the knowledge level of the user
- Emphasis the visual component
- Keep the user interested
- Incorporate both symbolic and iconic representations
- Include algorithm analysis and comparisons with other algorithm for the same problem
- Include execution history

The success of *Sorting Out Sorting* made sorting algorithms a perennial favorite for algorithm animation. Indeed, the sorting problem lends itself quite naturally to visual presentation via vertical or horizontal bars or sticks of different heights or lengths, which need to be rearranged according to their sizes (Figure 2.8). This presentation is convenient, however, only for illustrating actions of a typical sorting algorithm on small inputs. For larger files, *Sorting Out Sorting* used the ingenious idea of presenting data by a scatterplot of points on a coordinate plane, with the first coordinate representing an item's position in the file and the second one representing the item's value; with such a representation, the process of sorting looks like a transformation of a “random” scatterplot of points into the points along a frame's diagonal (Figure 2.9). In addition, most sorting algorithms work by comparing and exchanging two given items at a time—an event that can be animated relatively easily.



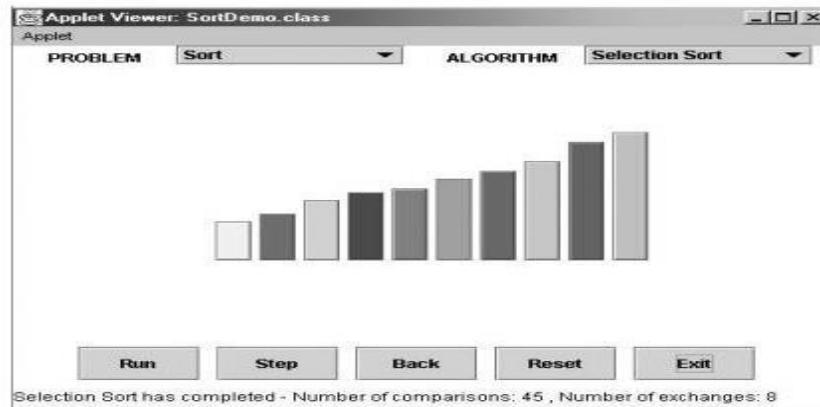


FIGURE 2.8 Initial and final screens of a typical visualization of a sorting algorithm using the bar representation.

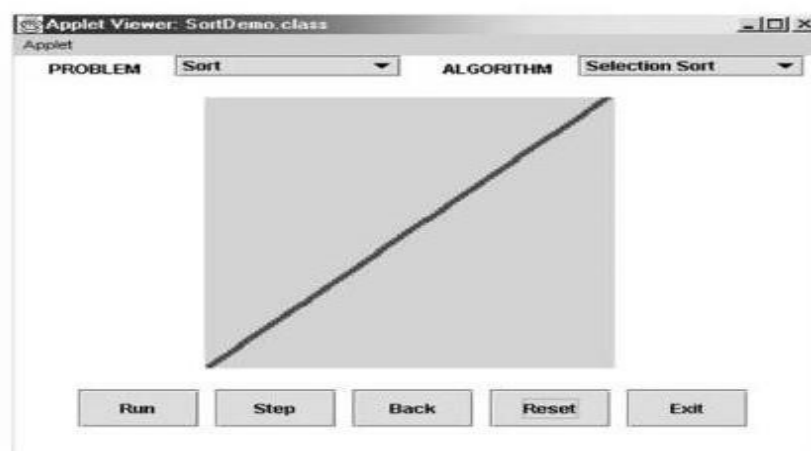
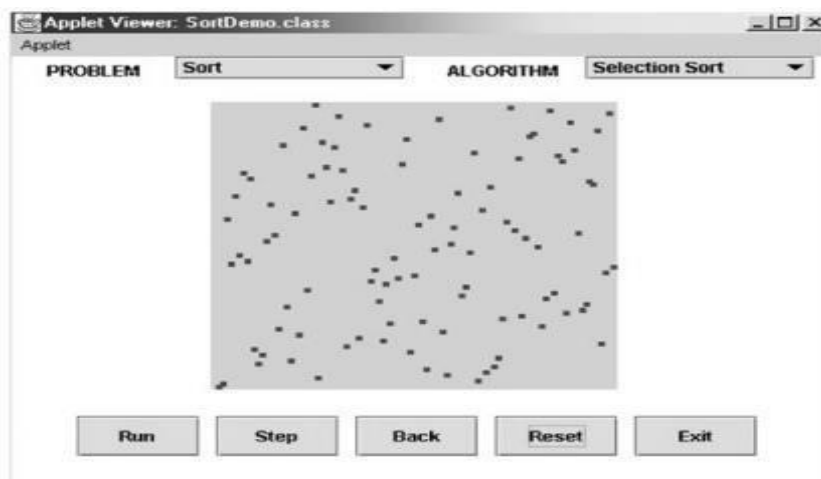


FIGURE 2.9 Initial and final screens of a typical visualization of a sorting algorithm using the scatterplot representation.



Applications:

1. Education - Seeks to help students learning algorithms.
2. Research - Helps to uncover some unknown features of algorithms.

IMPORTANT QUESTIONS

Part A

1. Show the notion of an algorithm. *Dec 2009 / May 2013*
2. What are six steps processes in algorithmic problem solving? *Dec 2009*
3. What is time and space complexity? *Dec 2012*
4. Define Algorithm validation. *Dec 2012*
5. Differentiate time complexity from space complexity. *May 2010*
6. What is a recurrence equation? *May 2010*
7. What do you mean by algorithm? *May 2013*
8. Define Big Oh Notation. *May 2013*
9. What is average case analysis? *May 2014*
10. Define program proving and program verification. *May 2014*
11. Define asymptotic notation. *May 2014*
12. What do you mean by recursive algorithm? *May 2014*
13. Establish the relation between O and Ω *Dec 2010*
14. If $f(n) = a_n n^m + \dots + a_1 n + a_0$. Prove that $f(n) = O(n^m)$. *Dec 2010*
15. Define the Fundamentals of Algorithmic Problem Solving
16. Short notes on Important Problem Types
17. Define Fundamentals of the Analysis of Algorithm Efficiency
18. Show the Analysis Framework
19. Define Asymptotic Notations and its properties
20. Define Mathematical analysis for Recursive and Non-recursive algorithms.

Part B

1. Explain the notion of algorithm. *May 2014*
2. Explain the fundamentals of algorithm. *May 2014*
3. Find the time complexity and space complexity of the following problems. Factorial using recursion and compute the nth Fibonacci number using iterative statements. *Dec 2012*
4. Solve the following recurrence relations: *Dec 2012*
 1. $T(n) = \begin{cases} 2T(n/2) + 3 & n > 2 \\ 2 & n = 2 \end{cases}$
 2. $T(n) = \begin{cases} 2T(n/2) + cn & n > 1 \\ a & n = 1 \end{cases}$ where a and c constants
5. Distinguish between Big Oh, Theta and Omega notation. *Dec 2012*
6. Analyse the best case, average and worst case analysis for linear search. *Dec 2012*
7. Explain how time complexity is calculated. Give an example. *Apr 2010*
8. Elaborate on asymptotic notation with example. *Apr 2010*
9. Briefly explain the time complexity, space complexity estimation *June 2013*
10. Write linear search algorithm and analyse its complexity. *June 2013*
11. Show the following equalities are correct *June 2013*
 - i. $5n^2 - 6n = \Theta(n^2)$
 - ii. $n! = O(n^n)$
 - iii. $n^3 + 10^6 n^2 = \Theta(n^3)$
 - iv. $2n^2 2^n + n \log n = \Theta(n^2 2^n)$
12. What are the features of an efficient algorithm? *June 2014*
13. What is space complexity? With an example explain the components of fixed and variable part in space complexity. *June 2014*
14. Explain towers of Hanoi problem and solve it using recursion. *June 2014*
15. Derive the recurrence relation for Fibonacci series algorithm : also carry out time complexity analysis. *June 2014*
16. Discuss in details about the efficiency of the algorithm with example. *Mar 2014*

17. Explain the procedure to calculate the time complexity of binary search using non-recursive Algorithm.
18. Explain briefly the time complexity and space complexity estimation. *Nov 2010*
19. Write a linear search algorithm and analyse its best, worst and average case time complexity.
20. Prove that for any two functions $f(n)$ and $g(n)$, we have $f(n) \rightarrow \Theta(g(n))$ if and only if $f(n) \rightarrow O(g(n))$ and $f(n) \rightarrow \Omega(g(n))$ *Nov 2010*
21. Explain the Mathematical analysis for non-recursive algorithm

ANNA UNIVERSITY APRIL/MAY 2015

PART-A

1. write algorithm to find the number of binary digits in the binary representation of a positive decimal integer **Part A – Refer Q. No. 56**
2. write down the properties of asymptotic notations. **Part A – Refer Q. No. 57**

PART-B

- 11.(a)if you have to solve the searching problem for a list of n numbers, how can you take advantage of the fact that the list is known to be sorted? Give separate answers for
- (i) List represented as arrays
- (ii) List represented as linked list Compare the time complexity involved in the analysis of both the algorithms **Refer Q. No. 27**

OR

- (b)(i)Derive the worst case analysis of merge sort using suitable illustration **Refer Q.No. 28**
- (ii) Derive a loose bound on the following equation:
 $F(x) = 35x^8 - 22x^7 + 14x^5 - 2x^4 - 4x^2 + x - 15$ **Q.No. 15**

ANNA UNIVERSITY NOV/DEC 2015

PART-A

1. The $(\log n)$ th smallest number of n unsorted numbers can be determined in $O(n)$ average-case time (True/False) **Refer Q. No. 60**
2. Fibonacci algorithm and its recurrence relation **Refer Q. No. 61**

PART-B

- 11.(a)(i)write Insertion sort algorithm and estimate its running time.(8) **Refer Q. No. 12**
- (ii)find the closest asymptotic tight bound by solving the recurrence equation
 $T(n) = 8T(n/2) + n^2$ with $(T(1) = 1)$ using recursion tree method.[Assume that $T(1) \in \Theta(1)$]
Refer Q. No. 14

OR

- (b)(i)Suppose W satisfies the following recurrence equation and base case (where c is a constant): $W(n) = c.n + W(n/2)$ and $W(1) = 1$. What is the asymptotic order of $W(n)$.
Refer Q. No. 14
- (ii)Show how to implement a stack using two queues. Analyze the running time of the stack Operations. **Refer Q. No. 13**

ANNA UNIVERSITY APRIL/MAY 2016

PART-A

1. Give the Euclid's algorithm for computing $\gcd(m, n)$ **Refer Q. No. 58**
2. Compare the order of growth $n(n-1)/2$ and n^2 . **Refer Q. No. 59**

PART-B

1. a.(i) Give the definition and Graphical Representation of O -Notation.(8) **Refer Q. No. 4**

- (ii) Give an algorithm to check whether all the Elements in a given array of n elements are distinct. Find the worst case complexity of the same. (8) **Refer Q. No.5(2)**

OR

- (b) Give the recursive algorithm which finds the number of binary digits in the binary representation of a positive decimal integer. Find the recurrence relation and complexity. (16) **Refer Q. No.6(3)**

ANNA UNIVERSITY NOV/DEC 2016

PART-A

1. Design an algorithm to compute the area and circumference of a circle **Refer Q. No. 63**
2. Define recurrence relation. **Refer Q. No. 45**

PART-B

- 11.(a)(i) Use the most appropriate notation to indicate the time efficiency class of sequential search algorithm in the worst case, best case and the average case. **Refer Q. No. 17**
- (ii) State the general plan for analyzing the time efficiency of nonrecursive algorithm and explain with an example (8) **Refer Q. No. 5**
- (b) Solve the recurrence relations **Refer Q. No. 16**

$$X(n) = x(n-1) + 5 \text{ for } n > 1 \quad x(1) = 0$$

$$X(n) = 3x(n-1) \text{ for } n > 1 \quad x(1) = 4$$

$$X(n) = x(n-1) + n \text{ for } n > 0 \quad x(0) = 0$$

$$X(n) = x(n/2) + n \text{ for } n > 1 \quad x(1) = 1 \text{ (solve for } n = 2^k)$$

$$X(n) = x(n/3) + 1 \text{ for } n > 1 \quad x(1) = 1 \text{ (solve for } n = 3^k) \text{ (16)}$$

ANNA UNIVERSITY APRIL/MAY 2017

PART-A

1. What is an algorithm? **Refer Q. No. 1**
2. Write an algorithm to compute the greatest common divisor of two numbers **Refer Q. No. 10**

PART-B

1. Explain briefly Big Oh notation, Omega notation and Theta notation give an example **Q. No. 30**
2. Briefly explain the mathematical analysis of recursive and non recursive algorithm **Q.No.35 & 40**

ANNA UNIVERSITY NOV/DEC 2017

PART-A

1. How to measure an algorithm's running time? **Refer Q. No. 21**
2. What do you mean by "worst case efficiency" of an algorithm. **Refer Q. No. 55**

PART-B

1. Discuss the steps in Mathematical analysis for recursive algorithms. Do the same for finding Factorial of a number **Refer Q. No. 6**
2. What are the Rules of Manipulate Big-Oh Expression and about the typical growth rates of algorithms? **Refer Q.No.4**

ANNA UNIVERSITY MAY/JUNE 2018

PART-A

1. Give the Euclid's algorithm for computing gcd of two numbers. **Refer Q. No. 58**
2. What is a basic operation? **Refer Q. No. 63**

PART-B

1. a) Define Big O notation, Big Omega and Big Theta Notation. Depict the same graphically and explain. **Refer Q.No.4**

b) Give the general plan for Analyzing the time efficiency of Recursive Algorithms and use recurrence to find number of moves for Towers of Hanoi problem. **Refer Q.No.6**

ANNA UNIVERSITY NOV/DEC 2018

PART-A

1. Define algorithm. List the desirable properties of an algorithm. **Refer Q. No. 64**
2. Define best, worst, average case time complexity. **Refer Q. No. 65**

PART-B

1. (i) Prove that if $g(n)$ is $\Omega(f(n))$ then $f(n)$ is $O(g(n))$. **Refer Q.No.18**
 (ii) Discuss various methods used for mathematical analysis of recursive algorithms. **Refer Q.No.6**
2. Write the asymptotic notations used for best case, average case and worst case analysis of algorithms. Write an algorithm for finding maximum element in an array. Give best, worst and average case complexities. **Refer Q.No.4**

ANNA UNIVERSITY APRIL/MAY 2019

PART-A

1. How do you measure the efficiency of an algorithm? - **Refer Q.No.29**
2. Prove that the of $f(n)=o(g(n))$ and $g(n)=o(f(n))$, then $f(n)=\theta g(n)$. - **Refer Q.No.66**

PART-B

- 1.a) (i) solve the following recurrence equation: - **Refer Q.No.8**
 1. $T(n)=T(n/2)+1$, where $n=2^k$ for all $k \geq 0$
 2. $T(n)= T(n/3)+ T(2n/3)+cn$, where 'c' is a constant and 'n' is the input size.
 - (ii) Explain the steps involved in problem solving. - **Refer Q.No.8**
- 2.(i) write an algorithm for determining the uniqueness of an array. Determine the time complexity of your algorithm. - **Refer Q.No.5**
 - (ii) Explain time-space trade off of the algorithm designed - **Refer Q.No.3**

ANNA UNIVERSITY NOV/DEC 2019

PART-A

1. State the transpose symmetry property of O and Ω - **Refer Q.No.66**
2. Define recursion - **Refer Q.No.67**

PART-B

1. a) i) Solve the following recurrence equations using iterative method or tree **Refer Q.No.6**
 ii) Elaborate asymptotic analysis of an algorithm with an example. **Refer Q.No.4**
2. b) write an algorithm using recursion that determines the GCD of two numbers. Determine the time and space complexity - **Refer Q.No.1.A**

ANNA UNIVERSITY NOV/DEC 2021

PART-A

1. Define algorithm with its properties. **Refer Q.No.1**
2. List the reasons for choosing an approximate algorithm. **Refer Q.No.68**

PART-B

1. a) i) Consider the problem of counting, in a given text the number of substrings that start with an A and end with a B. For example, there are four such substrings in CABAAXBYA. Design a brute-force algorithm for this problem and determine its efficiency class. **Refer Q.No.6**

ii) “The best-case analysis is not as important as the worst-case analysis of an algorithm”.
Yes or No ? Justify your answer with the help of an example. **Refer Q.No.11**

2. b) (i) Solve : $T(n) = 2T(n/2) + n^3$. **Refer Q.No.11**

(iii) Explain the importance of asymptotic analysis for running time of an algorithm with an example. **Refer Q.No.4**

ANNA UNIVERSITY NOV/DEC 2021

PART-A

1. Define the notation big-Omega. **Refer Q.No.14**
2. What is meant time complexity of an algorithm? **Refer Q.No.7**

PART-A

11. a) Outline worst case running time, best case running time and average case running time of an algorithm with an example?

b) Outline a recursive algorithm and non recursive algorithm with an example.

Refer Q.No.35 & 40



**DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING**

**COMMON FOR: DEPARTMENT OF INFORMATION
TECHNOLOGY**

CS8491 – COMPUTER ARCHITECTURE

YEAR / SEM: II / III

R – 2017

LECTURE NOTES

UNIT - I

COMPUTER ORGANIZATION & INSTRUCTIONS

INTRODUCTION

Computer architecture acts as the interface between the hardware and the lowest level software. Computer architecture refers to:

- Attributes of a system visible to programmers like datatype of variables.
- Attributes that have a direct impact on the execution of programs like clock cycle.

Computer Architecture is defined as study of the structure, behaviour, and design of computers.

Computer Organization: It refers to the operational units and their interconnections that realize the architectural specifications. It describes the function of and design of the various units of digital computer that store and process information. The attributes in computer organization refers to:

- Control signals
- Computer/peripheral interface
- Memory technology

Computer hardware: Consists of electronic circuits, displays, magnetic and optical storage media, electromechanical equipment and communication facilities.

Computer Architecture: It is concerned with the structure and behaviour of the computer. It includes the information formats, the instruction set and techniques for addressing memory. The attributes in computer architecture refers to the:

- Instruction set
- Data representation
- I/O mechanisms
- Addressing techniques

The basic **distinction between architecture and organization** is: the attributes of the former are visible to programmers whereas the attributes of the later describes how features are implemented in the system.

BASICS OF A COMPUTER SYSTEM

The modern day computer system's functional unit is given by Von Neumann Architecture.



Fig 1.1: Von Neumann Architecture

Input Unit

Computers accept the coded information through input unit. Computer must receive both data and program statements to function properly and must be able to solve problems. The method of feeding data and programs to a computer is accomplished by an input device. Input devices read data from a source, such as magnetic disks, and translate that data into electronic impulses for transfer into the CPU. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Central Processing Unit (CPU)

The CPU processes data transferred to it from one of the various input devices. It then transfers either an intermediate or final result of the CPU to one or more output devices. A central control section and work areas are required to perform calculations or manipulate data. The CPU is the computing center of the system. It consists of a control section, an arithmetic-logic section, and an internal storage section (memory unit). Each section within the CPU serves a specific function and has a particular relationship with the other sections within the CPU.

Memory Unit

It stores the programs and data. Memory unit is broadly classified into two types: Primary memory and Secondary memory.

1. Primary Memory:

It is a fast memory that operates at electronic speeds. Programs must be stored in the memory while they are being executed. The memory contains large no of semiconductor storage cells. Each cell carries 1 bit of information. The cells are processed in a group of fixed size called **Words**. To provide easy access to any word in a memory, a distinct address is associated with each word location. Addresses are numbers that identify successive locations. The number of bits in each word is called the **word length**. The word length ranges from 16 to 64 bits. There are 3 types of primary memory:

- I. RAM:** Memory in which any location can be reached in short and fixed amount of time after specifying its address is called RAM. Time required to access 1 word is called Memory Access Time.
- II. Cache Memory:** The small, fast, RAM units are called Cache. They are tightly coupled with processor to achieve high performance.
- III. Main Memory:** The largest and the slowest unit is the main memory.

Arithmetic & Logic Unit

Most computer operations are executed in ALU. The arithmetic-logic section performs arithmetic operations, such as addition, subtraction, multiplication, and division. Through internal logic capability, it tests various conditions encountered during processing and takes action based on the result. Data maybe transferred back and forth between these two sections several times before processing is completed. Access time to registers is faster than access time to the fastest cache unit in memory.

Output Unit

Its function is to send the processed results to the outside world.

Control Unit

The operations of Input unit, output unit, ALU are co-ordinate by the control unit. The control unit is the Nerve centre that sends control signals to other units and senses their states. The control section directs the flow of traffic (operations) and data. It also maintains order within the computer. The control section selects one program statement at a time from the program storage area, interprets the statement, and sends the appropriate electronic impulses to the arithmetic-logic and storage sections so they can carry out the instructions. The control section does not perform actual processing operations on the data.

The control section instructs the input device on when to start and stop transferring data to the input storage area. It also tells the output device when to start and stop receiving data from the output storage area. Data transfers between the processor and the memory are controlled by the control unit through **timing signals**. Information stored in the memory is fetched, under program control into an arithmetic and logic unit, where it is processed.

Evolution of Computers

- The word ‘computer’ is an old word that has changed its meaning several times in the last few centuries.
- Today, the word computer refers to computing devices, whether or not they are electronic, programmable, or capable of ‘storing and retrieving’ data.

The Mechanical Era (1623-1945)

- Wilhelm Schickhard, Blaise Pascal, and Gottfried Leibnitz were among mathematicians who designed and implemented calculators that were capable of addition, subtraction, multiplication, and division during the seventeenth century.
- The first multi-purpose or programmable computing device was probably **Charles Babbage’s Difference Engine**, which was begun in 1823 but never completed.
- In 1842, Babbage designed a more ambitious machine, called the **Analytical Engine** but unfortunately it also was only partially completed.
- Babbage, together with Ada Lovelace recognized several important programming techniques, including conditional branches, iterative loops and index variables.
- Babbage designed the machine which is the first to be used in computational science.
- In 1833, George Scheutz and his son, Edvard began work on a smaller version of the difference engine and by 1853 they had constructed a machine that could process 15-digit numbers and calculate fourth-order differences.
- The US Census Bureau was one of the first organizations to use the mechanical computers which used punch-card equipment designed by Herman Hollerith to tabulate data for the 1890 census.
- In 1911 Hollerith’s company merged with a competitor to found the corporation which in 1924 became **International Business Machines (IBM)**.

First Generation Electronic Computers (1937-1953)

- These devices used electronic switches, in the form of **vacuum tubes**, instead of electromechanical relays.
- The earliest attempt to build an electronic computer was by J. V. Atanasoff, a professor of physics and mathematics at Iowa State in 1937.
- Atanasoff set out to build a machine that would help his graduate students solve systems of partial differential equations.
- By 1941 he and graduate student Clifford Berry had succeeded in building a machine that could solve 29 simultaneous equations with 29 unknowns.
- However, the machine was not programmable, and was more of an electronic calculator.
- A second early electronic machine was Colossus, designed by Alan Turing for the British military in 1943.
- The first general purpose programmable electronic computer was the Electronic Numerical Integrator and Computer (**ENIAC**), built by J. Presper Eckert and John V. Mauchly at the University of Pennsylvania.
- ENIAC was controlled by a set of external switches and dials; to change the program required physically altering the settings on these controls.
- Research work began in 1943, funded by the Army Ordinance Department, which needed a way to compute ballistics during World War II.
- The machine was completed in 1945 and it was used extensively for calculations during the design of the hydrogen bomb.
- Eckert, Mauchly, and John von Neumann, a consultant to the ENIAC project, began work on a new machine before ENIAC was finished.
- The next development was **EDVAC**- Electronic Discrete Variable Computer.
- The main contribution of EDVAC, their new project, was the notion of a **stored program**.
- EDVAC was able to run orders of magnitude faster than ENIAC and by storing instructions in the same medium as data, designers could concentrate on improving the internal structure of the machine without worrying about matching it to the speed of an external control.

- Eckert and Mauchly later designed the first commercially successful computer, the **UNIVAC**(Universal Automatic Computer); in 1952.
- Software technology during this period was very primitive.
- The instructions were written in machine language that could be executed directly.

Second Generation (1954-1962)

- The second generation witnessed several important developments at all levels of computer system design, ranging from the technology used to build the basic circuits to the programming languages used to write scientific applications.
- Electronic switches in this era were based on **discrete diode and transistor technology** with a switching time of approximately 0.3 microseconds.
- The first machines to be built with this technology include **TRADIC** at Bell Laboratories in 1954 and TX-0 at MIT's Lincoln Laboratory.
- Index registers were designed for controlling loops and floating point units for calculations based on real numbers.
- A number of high level **programming languages** were introduced and these include FORTRAN (1956), ALGOL (1958), and COBOL (1959).
- **Batch processing systems** came to existence.
- Important commercial machines of this era include the IBM 704 and its successors, the 709 and 7094.
- In the 1950s the first two supercomputers were designed specifically for numeric processing in scientific applications.
- Multiprogrammed computers that serve many users concurrently came to existence. This is otherwise known as **time-sharing systems**.

Third Generation (1963-1972)

- Technology changes in this generation include the use of **integrated circuits**, or ICs.
- This generation led to the introduction of **semiconductor memories, microprogramming** as a technique for efficiently designing complex processors and the introduction of **operating systems** and time-sharing.

- The first ICs were based on small-scale integration (SSI) circuits, which had around 10 devices per circuit (or ‘chip’), and evolved to the use of medium-scale integrated (MSI) circuits, which had up to 100 devices per chip.
- Multilayered printed circuits were developed and core memory was replaced by faster, solid state memories.
- In 1964, Seymour Cray developed the CDC 6600, which was the first architecture to use **functional parallelism**.
- By using 10 separate functional units that could operate simultaneously and 32 independent memory banks, the CDC 6600 was able to attain a computation rate of one million floating point operations per second (Mflops).
- Five years later CDC released the 7600, also developed by Seymour Cray.
- The CDC 7600, with its pipelined functional units, is considered to be the first vector processor and was capable of executing at ten Mflops.
- The IBM 360/91, released during the same period, was roughly twice as fast as the CDC 660.
- Early in this third generation, Cambridge University and the University of London cooperated in the development of **CPL** (Combined Programming Language, 1963).
- CPL was an attempt to capture only the important features of the complicated and sophisticated ALGOL.
- However, like ALGOL, CPL was large with many features that were hard to learn.
- In an attempt at further simplification, Martin Richards of Cambridge developed a subset of CPL called **BCPL** (Basic Computer Programming Language, 1967).
- In 1970 Ken Thompson of Bell Labs developed yet another simplification of CPL called simply **B**, in connection with an early implementation of the UNIX operating system.

Fourth Generation (1972-1984)

- **Large scale integration** (LSI - 1000 devices per chip) and **very large scale integration** (VLSI - 100,000 devices per chip) were used in the construction of the fourth generation computers.
- Whole processors could now fit onto a single chip, and for simple systems the entire computer (processor, main memory, and I/O controllers) could fit on one chip.

- Gate delays dropped to about 1ns per gate. Core memories were replaced by semiconductor memories.
- Large main memories like CRAY 2 began to replace the older high speed vector processors, such as the CRAY 1, CRAY X-MP and CYBER.
- In 1972, Dennis Ritchie developed the **C language** from the design of the CPL and Thompson's B.
- Thompson and Ritchie then used C to write a version of UNIX for the DEC PDP-11.
- Other developments in software include very high level languages such as FP (functional programming) and Prolog (programming in logic).
- IBM worked with Microsoft during the 1980s to start what we can really call PC (Personal Computer) life today.
- IBM PC was introduced in October 1981 and it worked with the operating system (software) called 'Microsoft Disk Operating System (MS DOS) 1.0.
- Development of **MS DOS** began in October 1980 when IBM began searching the market for an operating system for the then proposed IBM PC and major contributors were Bill Gates, Paul Allen and Tim Paterson.
- In 1983, the **Microsoft Windows** was announced and this has witnessed several improvements and revision over the last twenty years.

Fifth Generation (1984-1990)

- This generation brought about the introduction of machines with hundreds of processors that could all be working on different parts of a single program.
- The scale of integration in semiconductors continued at a great pace and by 1990 it was possible to build chips with a million components - and semiconductor memories became standard on all computers.
- Computer networks and single-user workstations also became popular. Parallel processing started in this generation.
- The Sequent Balance 8000 connected up to 20 processors to a single shared memory module though each processor had its own local cache.
- The machine was designed to compete with the **DEC VAX-780** as a general purpose Unix system, with each processor working on a different user's job.

- However Sequent provided a library of subroutines that would allow programmers to write programs that would use more than one processor, and the machine was widely used to explore parallel algorithms and programming techniques.
- The **Intel iPSC-1**, also known as ‘the hypercube’ connected each processor to its own memory and used a network interface to connect processors.
- This distributed memory architecture meant memory was no longer a problem and large systems with more processors (as many as 128) could be built.
- Also introduced was a machine, known as a **data-parallel** or SIMD where there were several thousand very simple processors which work under the direction of a single control unit.
- Both wide area network (WAN) and local area network (LAN) technology developed rapidly.

Sixth Generation (1990 -)

- Most of the developments in computer systems since 1990 have not been fundamental changes but have been gradual improvements over established systems.
- This generation brought about gains in parallel computing in both the hardware and in improved understanding of how to develop algorithms to exploit parallel architectures.
- Workstation technology continued to improve, with processor designs now using a combination of RISC, pipelining, and parallel processing.
- Wide area networks, network bandwidth and speed of operation and networking capabilities have kept developing tremendously.
- Personal computers (PCs) now operate with Gigabit per second processors, multi-Gigabyte disks, hundreds of Mbytes of RAM, colour printers, high-resolution graphic monitors, stereo sound cards and graphical user interfaces.
- Thousands of software (operating systems and application software) are existing today and Microsoft Inc. has been a major contributor. Microsoft is said to be one of the biggest companies ever, and its chairman – Bill Gates has been rated as the richest man for several years.

- Finally, this generation has brought about **micro controller technology**. Micro controllers are 'embedded' inside some other devices so that they can control the features or actions of the product.
- They work as small computers inside devices and now serve as essential components in most machines.

Great Ideas in Computer Architecture

The ideas that marked tremendous improvement in the field of computer architecture are briefly discussed here.

1. Moore's Law

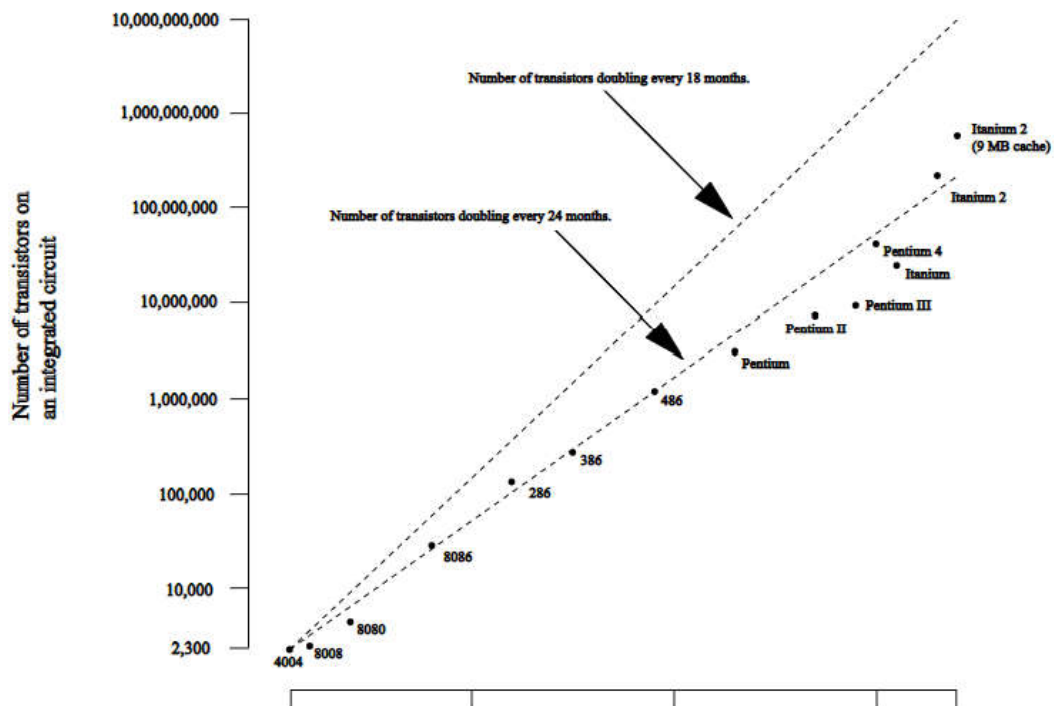


Fig 1.2: Illustration of Moore's Law

Moore's law states that the number of transistors will double every 18 months.

It is an observation that the number of transistors in a dense integrated circuit doubles about every two years. It is an observation and projection of a historical trend and not a physical or natural law.

2. Abstract Design

It is a major productivity technique for hardware and software. Abstractions are used to represent the design at different levels of representation. The detailed lower-level design details from the higher levels.

3. Performance through parallelism

Parallelism executes programs faster by performing several computations at the same time. This requires hardware with multiple processing units. The overall performance of the system is significantly increased by performing operations in parallel.

4. Performance through Pipelining

Pipelining increases the CPU instruction throughput. **Throughput** is a performance metric which is the number of instructions completed per unit of time. But it does not reduce the execution time of an individual instruction. It increases the execution time of each instruction due to overhead in the pipeline control. The increase in instruction throughput means that a program runs faster and has lower total execution time.

5. Make the Common Case Fast

Making the common case fast will tend to enhance performance better than optimizing the rare case. Ironically, the common case is often simpler than the rare case and hence is often easier to enhance. In making a design trade-off, favor the frequent case over the infrequent case. Amdahl's Law can be used to quantify this principle. This also applies when determining how to spend resources, since the impact on making some occurrence faster is higher if the occurrence is frequent. This will:

- Helps performance
- Is simpler and can be done faster

6. Performance via prediction

The computer can perform better (on average) by making rational guesses on the decisions. Instead of wasting clock cycles for certain results, the computers can remarkably improve the performance

7. Hierarchy of memories

Programmers want memory to be fast, large, and cheap. The memory speed is a primary factor in determining the performance of the system. The memory capacity limits the size of problems that can be solved.

Architects have found that hierarchy of memories will be a solution for all these issues. The fastest, smallest, and most expensive memory per bit is placed the top of the hierarchy and the slowest, largest, and cheapest per bit is at the bottom. **Caches** give the illusion that main memory is nearly as fast as the top of the hierarchy and nearly as big and cheap as the bottom of the hierarchy.

8. Dependability via Redundancy

Computers need to be fast and dependable. Since any physical device can fail, we make systems dependable by including redundant components that can take over when a failure occurs and help detect failures. Restoring the state of the system is done by redundancy.

Technologies

Up until the early 1970's computers used magnetic core memory, which was slow, cumbersome, and expensive and thus appeared in limited quantities. The situation improved with the introduction of transistor-based dynamic random-access memory (DRAM, invented at IBM in 1966) and static random-access memory (SRAM). A **transistor** is simply an on/off switch controlled by electricity. The **integrated circuit (IC)** combined dozens to hundreds of transistors into a simple chip. **Very large-scale integrated (VLSI)** circuit is a device containing hundreds of thousands to millions of transistors.

Manufacturing of IC:

Integrated circuits are chips manufactured on silicon wafers. Transistors are placed on wafers through a chemical etching process. Each wafer is cut into chips which are packed individually.

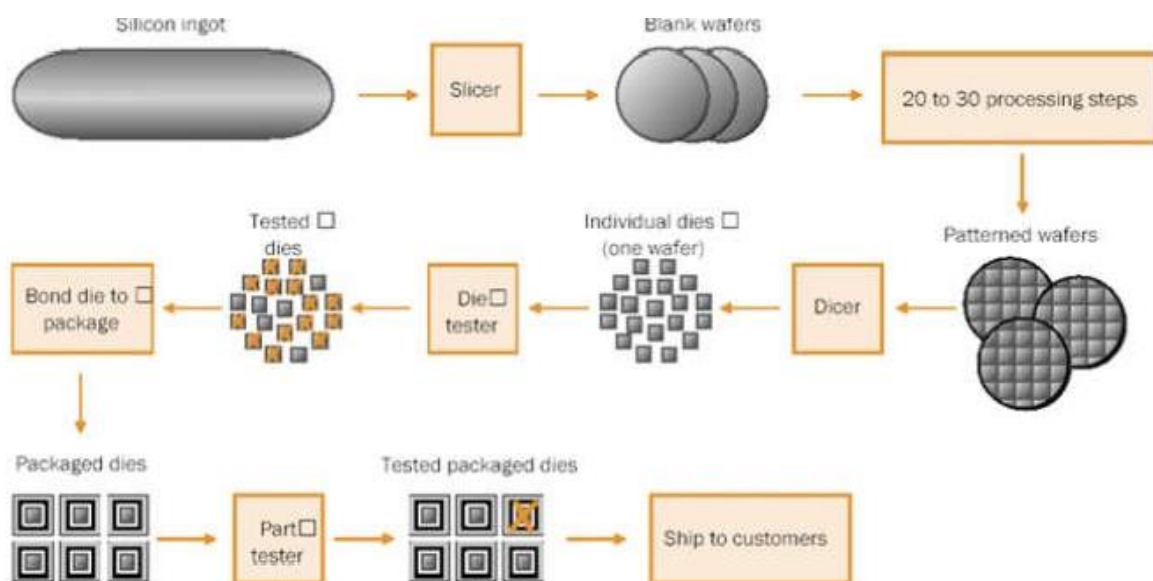


Fig 1.3: Chip manufacturing process

After being sliced from the silicon ingot, blank wafers are put through 20 to 40 steps to create patterned wafers. These patterned wafers are then tested with a wafer tester, and a map of the good parts is made. Then, the wafers are diced into dies. The good dies are then bonded into packages and tested one more time before shipping the packaged parts to customers.

Cost of an IC is found from:

1. Cost per die = (cost per wafer) / ((dies per wafer) * yield)

Yield refers the fraction of dies that pass testing.

2. Dies / wafer = wafer area / die area

3. Yield = $1 / (1 + (\text{defects per area} * \text{die area}) / 2)^2$

Programmable Logic Device (PLD)

A programmable logic device (PLD) is an electronic component used to build reconfigurable digital circuits. Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed, that is, reconfigured.

The major limitations of PLD:

- Consume space due to large number of switches for programmability
- Low speed due to the presence of many switches.

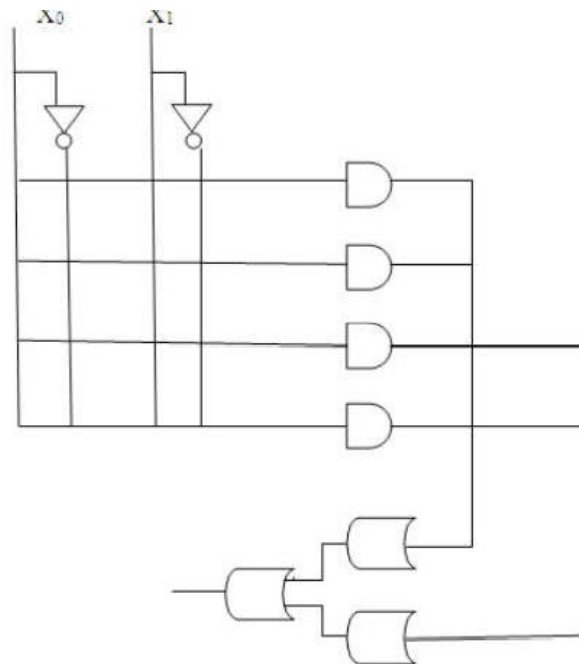


Fig 1.4: Programmable Logic Device

Custom chips

An Application-Specific Integrated Circuit (ASIC) is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use. Application-Specific Standard Products (ASSPs) are intermediate between ASICs and industry standard integrated circuits.

Performance

Elapsed time and throughput are two different ways of measuring speed.

- **Elapsed time** or wall-clock time or response time is the total time to complete a task, including disk accesses, memory accesses, input/output (I/O) activities, operating system overhead. It is the better measure for processor speed because it is less dependent on other system components.
- **CPU execution time** is the actual time the CPU spends computing for a specific task.
- The **User CPU time** is the CPU time spent in a program itself. **System CPU time** is the CPU time spent in the operating system performing tasks on behalf of the program.
- The **CPU Performance** equation (CPU Time) is the product of number of instructions executed, Average CPI of the program and CPU clock cycle.

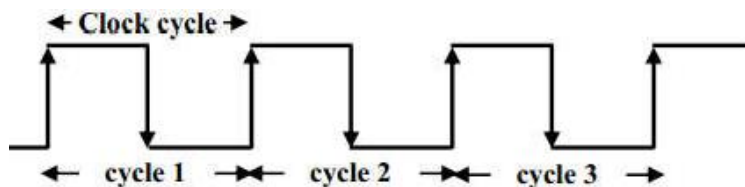
$$CPU\ Time = \frac{Seconds}{Pr\ ogram} = \frac{Instructions}{Pr\ ogram} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle}$$

- Performance is inversely proportional to execution time. Performance ratios are inverted from time ratios.

$$Performance\ improvement\ ratio = \frac{Performance\ after\ change}{Performance\ before\ change} = \frac{Execution\ time\ before\ change}{Execution\ time\ after\ change}$$

- Clock cycle is the time for one clock period, usually of the processor clock, which runs at a constant rate.
- Clock period is the length of each clock cycle.
- The CPU clock rate depends on CPU organisation and hardware implementation.

$$Clock\ Rate = \frac{1}{Clock\ Cycle}$$



- **Cycles Per Instruction (CPI)** is count of clock cycles taken by an instruction to complete its execution.

$$InstructionsPerCycle(IPC) = \frac{1}{CyclesPerInstruction}$$

- Performance is improved by reducing number of clock cycles, increasing clock rate and hardware designer must often trade off clock rate against cycle count.
- Workload is a set of programs run on a computer that is either the actual collection of applications run by a user or is constructed from real programs to approximate such a mix. A typical workload specifies both the programs as well as the relative frequencies.
- To evaluate two computer systems, a user would simply compare the execution time of the workload on the two computers.
- Alternatively, set of benchmarks containing several typical engineering or scientific applications can be used. A CPU benchmark (CPU benchmarking) is a series of tests designed to measure the performance of a computer or device CPU. A set of standards, or baseline measurements are used to compare the performance of different systems, using the same methods and circumstances.

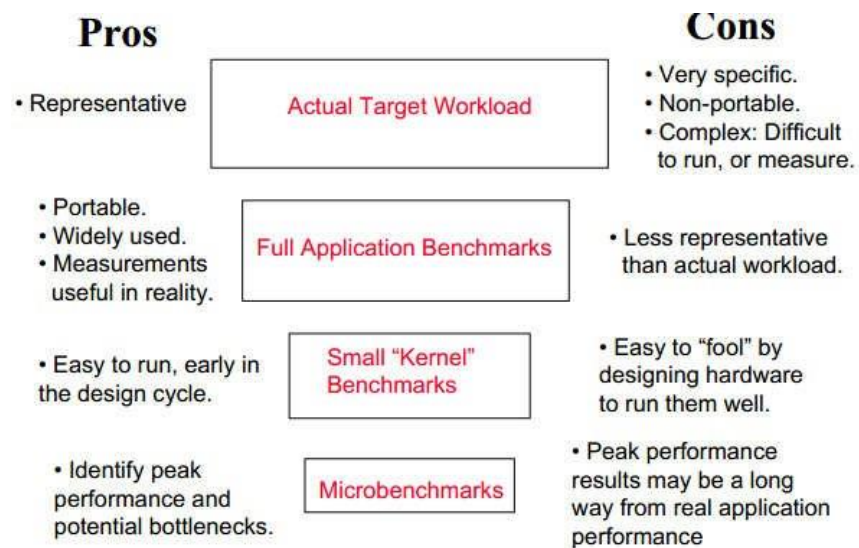


Fig 1.6: Types of Benchmark Programs

- The use of benchmarks whose performance depends on very small code segments encourages optimizations in either the architecture or compiler that target these segments.
- The arithmetic mean is proportional to execution time, assuming that the programs in the workload are each run an equal number of times.
- **Weighted arithmetic mean** is an average of the execution time of a workload with weighting factors designed to reflect the presence of the programs in a workload; computed as the sum of the products of weight.

Example 1.1: For a given program, the execution time on machine A is 1s and on B is 10s. Find the performance or speed up of the machines.

Execution_A = 1s

Execution_B = 10s

$$\text{Speedup} = \frac{\text{Performance of A}}{\text{Performance of B}} = \frac{\text{Execution of B}}{\text{Execution of A}}$$

$$\text{Speedup} = 10/1 = 10$$

The performance of machine A is 10 times faster than that of B.

Example 1.2: For a certain program with 1,00,00,000 instructions, find the execution time given the average CPI is 2.5 cycles/instruction and clock rate as 200MHz.

Number of instructions = 1,00,00,000

Average CPI = 2.5 cycles/instruction

Clock rate = 200MHz = 200,000,000 Hz

Clock cycle = 1/Clock rate = 1/200,000,000 = 5 x 10⁻⁹s

$$\text{CPU Time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

CPU Time = 100,000,000 x 2.5 x 5 x 10⁻⁹

= 0.125 s

Example 1.3: For a certain program with 1,00,00,000 instructions has an average CPI is 2.5 cycles/instruction and clock rate as 200MHz. When a new optimization compiler is deployed, the instruction count was reduced to 95,00,000 with new CPI=3.0 cycles/instruction at modified clock rate of 300MHz. Find the speedup.

$$\begin{aligned} \text{Speedup} &= \frac{\text{OldExecutionTime}}{\text{NewExecutionTime}} = \frac{I_{old} \times \text{CPI}_{old} \times \text{Clockcycle}_{old}}{I_{new} \times \text{CPI}_{new} \times \text{ClockCycle}_{new}} \\ &= (10000000 \times 2.5 \times 5 \times 10^{-9}) / (9500000 \times 3 \times 3.33 \times 10^{-9}) \\ &= 1.315 \end{aligned}$$

The new compiler is 1.315 times faster than the old one.

Example 1.4: A program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build a computer, B, which with run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 time as many clock cycles as computer A for this program. What clack rate should we tell the designer to target?

$$\begin{aligned} \text{Clock rate of B} &= \text{ClockCycles}_B / \text{CPUTime}_B \\ &= 1.2 \times \text{ClockCycles}_A / 6 \end{aligned}$$

$$\begin{aligned} \text{ClockCycles}_A &= \text{CPU Time}_A \times \text{ClockRate}_A \\ &= 10 \times 2 = 20 \times 10^9 \end{aligned}$$

$$\begin{aligned} \text{ClockCycles}_B &= 1.2 \times 20 \times 10^9 / 6 \\ &= 4 \text{ GHz} \end{aligned}$$

Example 1.5: Suppose we have two implementations of the same instruction set architecture. Computer A has a clock cycle time of 250ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

Computer A: Cycle Time = 250ps, CPI = 2.0

Computer B: Cycle Time = 500ps, CPI = 1.2

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$

$$= I \times 2.0 \times 250 = I \times 500$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$

$$= I \times 1.2 \times 500 = I \times 600$$

$$\frac{\text{CPUtime}_B}{\text{CPUtime}_A} = 1.2$$

Powerwall

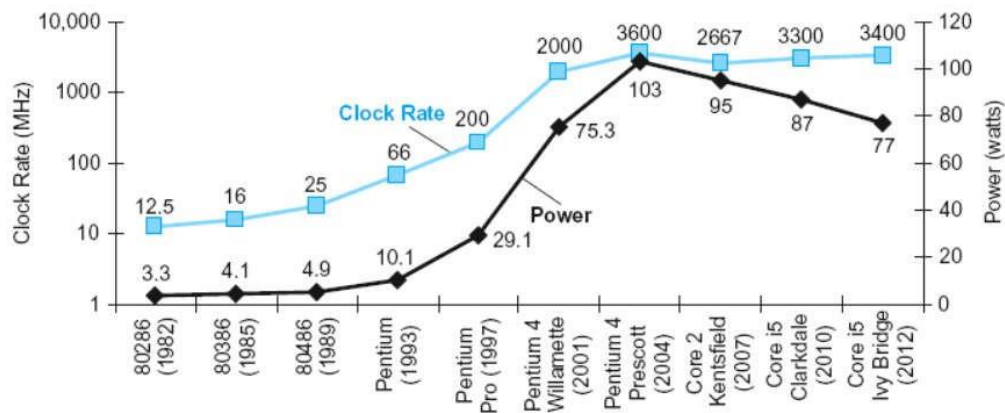


Fig 1.7: Clock rate and Power

- **Power wall** refers to the representational wall signifying the peak power constraint of a system.
- Clock rate and Power for Intel x86 microprocessors over eight generations and 25 years is shown in Fig 1.7.
- The Pentium 4 made a dramatic jump in clock rate and power but less so in performance.
- The Prescott thermal problems led to the abandonment of the Pentium 4 line. The Core 2 line reverts to a simpler pipeline with lower clock rates and multiple processors per chip.
- Continuous technology scaling like reduction of the transistor feature sizes makes it possible to pack more transistors in a given chip die area.
- Reduced supply voltage, simultaneous switching of these transistor devices causes a tremendous increase in the power density, leading to the power wall disaster.

- An increase in the power density increases the chip temperature, which slows down the transistor switching rate and hence, the overall speed of the computer.
- Cooling solutions are very expensive, and hence, computer architects have focused on innovating device, circuit and architecture level techniques to combat power wall.
- **Dynamic voltage and frequency scaling** are solutions for these problems. Here the operating voltage and frequency of the chip are dynamically controlled based on the chip activity.
- In CMOS (complementary metal oxide semiconductor) IC technology

$$\text{Power} = \text{Capacitiveload} \times \text{Voltage}^2 \times \text{Frequency}$$

Example 1.6: Suppose we developed a new, simpler processor that has 85% of the capacitive load of the more complex older processor. Further, assume that it has adjustable voltage so that it can reduce voltage 15% compared to processor B, which results in 15% shrink in frequency. What is the impact on dynamic power? Given : 85% of capacitive load of old CPU, 15% voltage reduction, 15% frequency reduction

$$\frac{P_{\text{new}}}{P_{\text{old}}} = \frac{(C_{\text{old}} \times 0.85) \times (V_{\text{old}} \times 0.85)^2 \times (F_{\text{old}} \times 0.85)}{C_{\text{old}} \times V_{\text{old}}^2 \times F_{\text{old}}} = 0.85^4 = 0.52$$

The new processor uses 0.52 the power of the old processor.

From Uniprocessors to Multiprocessors

The performance of the computers has drastically increased when the technology has drifted from uniprocessor systems to multiprocessor system. As the core computing units were made more powerful, the performance of the processors also increased significantly.

Uniprocessor system is a type of architecture that is based on a single computing unit. All the operations were done sequentially on the same unit. Multiprocessor systems are based on executing instructions on multiple computing units.

The multiprocessor architectures, is based on Flynn Taxonomy.

Flynn's taxonomy is a classification of parallel computer architectures that are based on the number of concurrent instruction and data streams available in the architecture.

Single Instruction, Single Data (SISD):

- This is a uniprocessor machine which is capable of executing a single instruction, operating on a single data stream.
- The machine instructions are processed in a sequential manner and computers adopting this model are popularly called sequential computers.
- Most conventional computers have SISD architecture.
- All the instructions and data to be processed have to be stored in primary memory.
- The speed of the processing element in the SISD model is limited by the rate at which the computer can transfer information internally.

Multiple Instruction, Single Data (MISD):

- An MISD computing system is a multiprocessor machine capable of executing different instructions on different Processing Elements but all of them operating on the same dataset.

Single Instruction, Multiple Data (SIMD):

- This machine capable of executing the same instruction on all the CPUs but operating on different data streams.
- Machines based on an SIMD model are well suited to scientific computing since they involve lots of vector and matrix operations. So that the information can be passed to all the Processing Elements (PEs) organized data elements of vectors can be divided into multiple sets and each PE can process one data set.

Multiple Instruction, Multiple Data (MIMD):

- This is capable of executing multiple instructions on multiple data sets.
- Each PE in the MIMD model has separate instruction and data streams; therefore machines built using this model are capable to any kind of application.
- Unlike SIMD and MISD machines, PEs in MIMD machines work asynchronously.

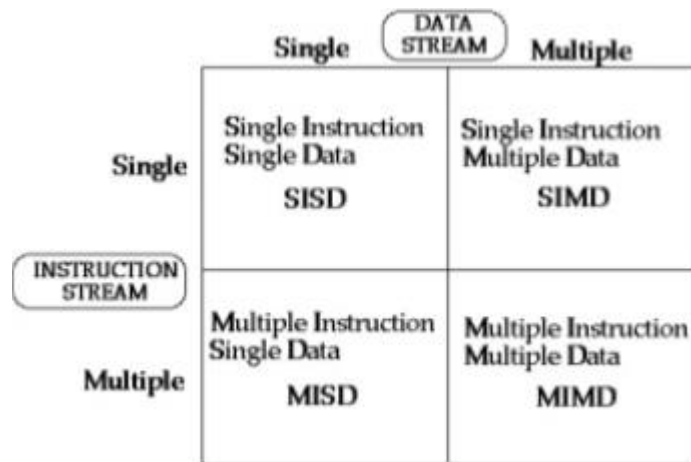


Fig 1.8: Flynn's Taxonomy

Apart from these architectures, MIPS Technologies developed a Microprocessor without Interlocked Pipeline Stages on Reduced Instruction Set Computer (RISC).

Concern for Power

- The power limit has forced a dramatic change in the design of microprocessors. Since 2002, the rate has slowed from a factor of 1.5 per year to a factor of 1.2 per year.
- Most of the desktop manufacturing companies are shipping microprocessors with multiple processors per chip, where the benefit increased throughput than on response time. This is done at the cost of increase in power.
- To reduce confusion between the words processor and microprocessor, companies refer to processors as cores and such microprocessors are generically called **multicore microprocessors**.
- A **quadcore** microprocessor is a chip that contains four processors or four cores.
- In the past, programmers could rely on innovations in hardware, architecture, and compilers to double performance of their programs every 18 months without having to change a line of code.
- Today, for programmers to get significant improvement in response time, they need to rewrite their programs to take advantage of multiple processors.

- Moreover, to get the historic benefit of running faster on new microprocessors, programmers will have to continue to improve performance of their code as the number of cores increases.

ADDRESSING AND ADDRESSING MODES

Each instruction of a computer specifies an operation on certain data.

The different ways in which the location of an operand is specified in an instruction is called as Addressing mode.

Different operands will use different addressing modes. One or more bits in the instruction format can be used as mode field. The value of the mode field determines which addressing mode is to be used. The effective address will be either main memory address or a register.

The most common addressing modes are:

1. Immediate addressing mode
2. Direct addressing mode
3. Indirect addressing mode
4. Register addressing mode
5. Register indirect addressing mode
6. Displacement addressing mode
7. Stack addressing mode

1. Immediate Addressing:

- This is the simplest form of addressing. Here, the operand is given in the instruction.
- This mode is used to define constant or set initial values of variables.
- The advantage of this mode is that no memory reference other than instruction fetch is required to obtain operand.
- The disadvantage is that the size of the number is limited to the size of the address field because most instruction sets are small compared to word length.

- **Example:** ADD 3
- Adds 3 to contents of accumulator and 3 is the operand.

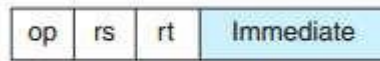


Fig 1.9: Immediate Mode

2. Direct Addressing:

- In direct addressing mode, effective address of the operand is given in the address field of the instruction.
- It requires one memory reference to read the operand from the given location and provides only a limited address space.
- Length of the address field is usually less than the word length.
- **Example :** Move P, Ro
Add Q, Ro

Where P and Q are the address of operand, R_0 is any register. Sometimes Accumulator(AC) is the default register. Then the instruction will look like:
Add A

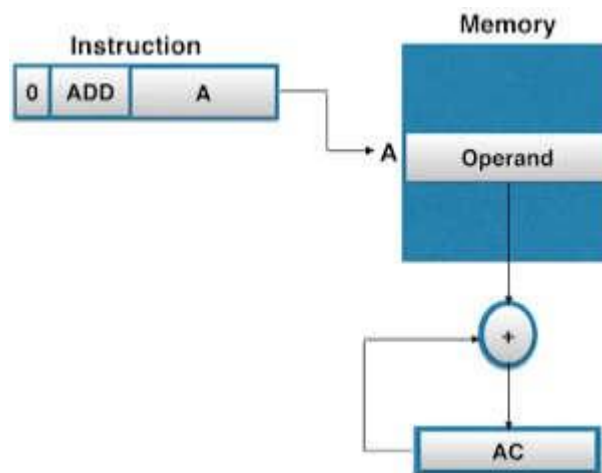


Fig 1.10: Direct Addressing modes

3. Indirect or Pseudodirect Addressing:

- Indirect addressing mode, the address field of the instruction refers to the address of a word in memory, which in turn contains the full length address of the operand.
- The address field of instruction gives the memory address where on, the operand is stored in memory.
- Control fetches the instruction from memory and then uses its address part to access memory again to read Effective Address.
- The advantage of this mode is that for the word length of N, an address space of $2N$ can be addressed.
- The disadvantage is that instruction execution requires two memory references to fetch the operand.
- Multilevel or cascaded indirect addressing can also be used.
- **Example:** Effective Address (EA) = (A).
- The operand will be present in the memory location A.

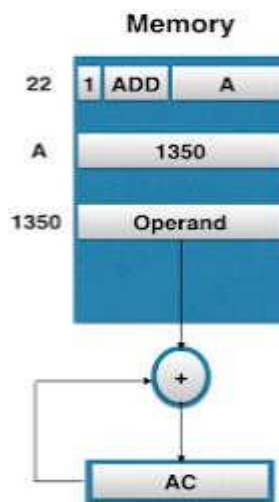


Fig 1.11: Indirect Addressing Modes

4. Register Addressing:

- Register addressing mode is similar to direct addressing. The only difference is that the address field of the instruction refers to a register rather than a memory location.

- 3 or 4 bits are used as address field in the instruction to refer 8 to 16 general purpose registers (GPR).
- The operands are in registers that reside within the CPU.
- The instruction specifies a register in CPU, which contain the operand.
- There is no need to compute the actual address as the operand is in a register and to get operand there is no memory access involved.
- The advantages of register addressing are small address field is needed in the instruction and faster instruction fetch.
- The disadvantages include very limited address space and usage of multiple registers helps in performance but it complicates the instructions.
- **Example:** MOV AX, BX

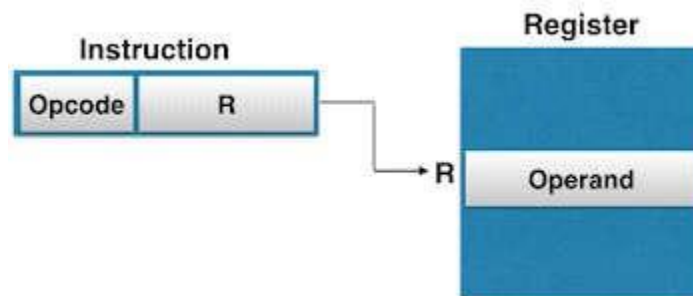


Fig 1.12: Register Mode

5. Register Indirect Addressing:

- This mode is similar to indirect addressing. The address field of the instruction refers to a register.
- The instruction specifies a register in CPU whose contents give the operand in memory.
- The selected register contains the address of operand rather than the operand itself.
- The register contains the effective address of the operand. This mode uses one memory reference to obtain the operand.

- Control fetches instruction from memory and then uses its address to access Register and looks in Register(R) for effective address of operand in memory.
- The address space is limited to the width of the registers available to store the effective address.
- **Example:MOV AL, [BX]**

Code example in Register:

MOV BX, 1000H

MOV 1000H, operand

- The instruction(MOV AL, [BX]) specifies a register[BX] which contain the address of operand(1000H) rather than address itself.

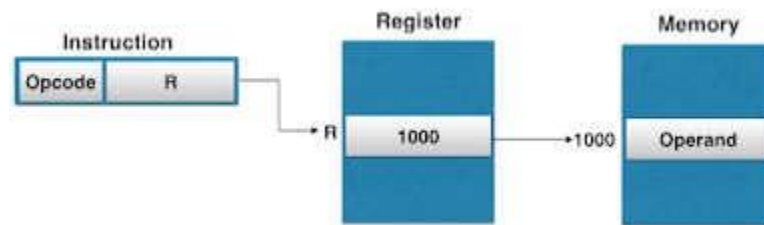


Fig 1.13: Register Indirect Mode

6. Displacement Addressing:

- It is a combination of direct addressing or register indirect addressing mode.
- Displacement Addressing Modes requires that the instruction have two address fields, at least one of which is explicit means, one is address field indicate direct address and other indicate indirect address.
- Value contained in one addressing field is A, which is used directly and the value in other address field is R, which refers to a register whose contents are to be added to produce effective address.
- **Example: EA=A+(R)**

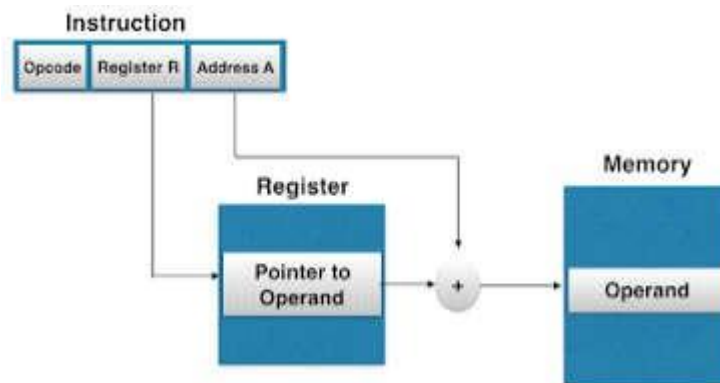


Fig 1.14 a): Displacement Addressing Modes

- In displacement addressing mode there are 3 types of addressing mode.
- **Relative addressing:**

The contents of program counter is added to the address part of instruction to obtain the Effective Address. The address field of the instruction is added to implicitly reference register Program Counter to obtain effective address.

Example: $EA=A+PC$

Assume that PC contains the value 825 and the address part of instruction contain the value 24, then the instruction at location 825 is read from memory during fetch phase and the Program Counter is then incremented by one to 826. Here both PC and instruction contains address. The effective address computation for relative address mode is $826+24=850$

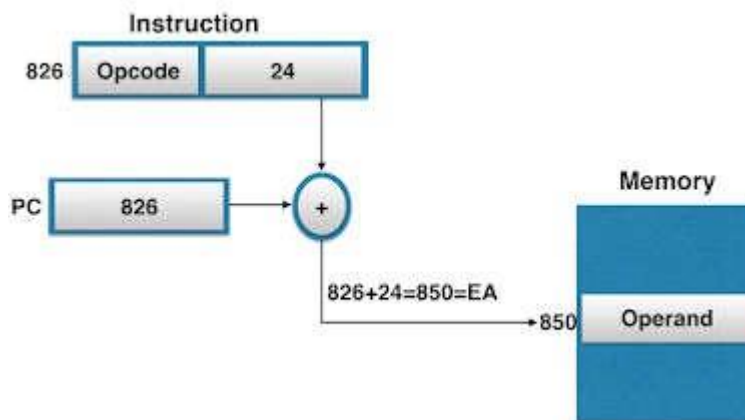


Fig 1.14 b): Relative addressing

- **Base register addressing**

The content of the Base Register is added to the direct address part of the instruction to obtain the effective address. The address field point to the Base Register and to obtain EA, the contents of Instruction Register, is added to direct address part of the instruction. This is similar to indexed addressing mode except that the register is now called as Base Register instead of Index Register.

Example: $EA = A + \text{Base}$

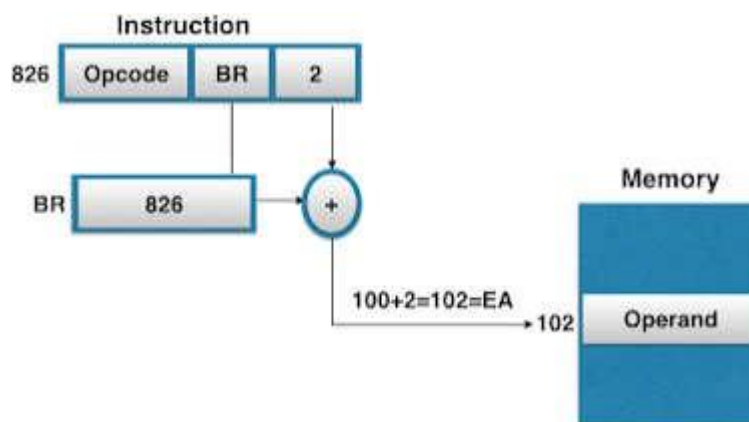


Fig 1.14 c): Base Register Addressing Mode

- **Indexed addressing:**

The content of Index Register is added to direct address part of instruction to obtain the effective address. The register indirect addressing field of instruction point to Index Register, which is a special CPU register that contain an Indexed value, and direct addressing field contain base address.

The data array is in memory and each operand in the array is stored in memory relative to base address. The distance between the beginning address and the address of operand is the indexed value stored in indexed register.

Any operand in the array can be accessed with the same instruction, which provided that the index register contains the correct index value i.e., the index register can be incremented to facilitate access to consecutive operands.

Example: $EA = A + \text{Index}$

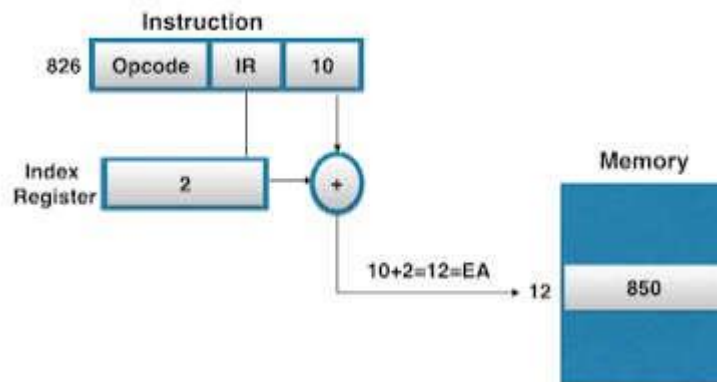


Fig 1.14d): Indexed Addressing

7. Stack Addressing:

- Stack is a linear array of locations referred to as last-in first out queue.
- The stack is a reserved block of location, appended or deleted only at the top of the stack.
- Stack pointer is a register which stores the address of top of stack location.
- This mode of addressing is also known as **implicit addressing**.
- Example: Add
- This instruction pops two items from the stack and adds.

Additional Modes:

There are two additional modes. They are:

- Auto-increment mode
- Auto-decrement mode

These are similar to Register indirect Addressing Mode except that the register is incremented or decremented after(or before) its value is used to access memory. These modes are required because when the address stored in register refers to a table of data in memory, then it is necessary to increment or decrement the register after every access to table so that next value is accessed from memory.

Auto-increment mode:

- Auto-increment Addressing Mode are similar to Register Indirect Addressing Mode except that the register is incremented after its value is loaded (or accessed) at another location like accumulator(AC).
- The Effective Address of the operand is the contents of a register in the instruction.
- After accessing the operand, the contents of this register is automatically incremented to point to the next item in the list.
- **Example:** (R) +.
- The contents in register R will be accessed and then it will be incremented to point to the next item in the list.

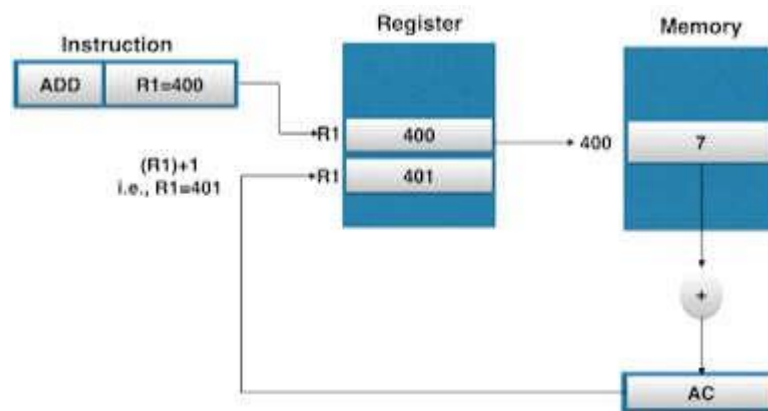


Fig 1.16: Auto-increment Mode

- The effective address is (R)=400 and operand in AC is 7. After loading R1 is incremented by 1, it becomes 401.

Auto-decrement mode:

- Auto-decrement Addressing Mode is reverse of auto-increment, as in it the register is decremented before the execution of the instruction.
- Effective address is equal to $EA=(R) - 1$
- The Effective Address of the operand is the contents of a register in the instruction.
- After accessing the operand, the contents of this register is automatically decremented to point to the next item in the list.

- **Example:** - (R)
- The contents in register R will be decremented and then it is accessed.

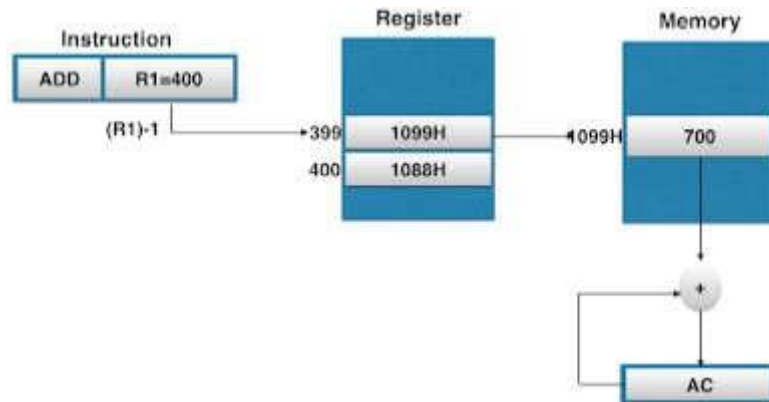


Fig 1.17: Auto Decrement Addressing Mode

INSTRUCTIONS

An instruction is a binary code, which specifies a basic operation for the computer.

- Operation Code (opcode) defines the operation type. Operands define the operation source and destination.
- **Instruction Set Architecture (ISA)** describes the processor in terms of what the assembly language programmer sees, i.e. the instructions and registers.
- The opcodes and operands follows Stores Program Concept.

Stored Program Concept is an idea that instructions and data of many types can be stored in memory as numbers, leading to the stored program computer.

Operations

- The computer performs the arithmetic through operations.
- The MIPS arithmetic instruction performs only one operation and must always have exactly three variables.

Example: Add a, b, c

Adds b and c and stores the sum in a.

- The hardware for a variable number of operands is more complicated than hardware for a fixed number.
- It is always essential to design the instructions with same number of operands so as to simplify the hardware requirement.

Operands

- The operands of arithmetic instruction must be from specially built memory locations called **registers**.
- The registers are accessed as 32 bit groups termed as **words**. MIPS architecture supports 32 registers.

Memory Operands

- The operands are always stored in registers.
- Data transfer instruction is a command that moves data between memory and registers.
- Address of an operand is a value used to delineate the location of a specific data element within a memory array.
- The data transfer instruction that copies data from memory to a register is traditionally called **load (lw- load word)**.
- The format of the load instruction is the name of the operation followed by the register to be loaded, then a constant and register used to access memory.
- The sum of the constant portion of the instruction and the contents of the second register forms the memory address.
- **Store (sw- store word)** instruction copies data from a register to memory.
- The format of a store is the name of the operation, followed by the register to be stored, then offset to select the array element, and finally the base register.
- The MIPS address is specified in part by a constant and in part by the contents of a register.

- Many programs have more variables than computers have registers. The compiler tries to keep the most frequently used variables in registers and places the rest in memory, using loads and stores to move variables between registers and memory.
- The process of putting less commonly used variables into memory is called **spilling registers**.

Constant or Immediate Operands

- Sometimes it is necessary to load a constant from memory to use one. The constants would have been placed in memory when the program was loaded.

Example: `addi $s3,$s3,10`

- This instruction is interpreted as addition of content of \$s3 and the value 10. The sum is stored in \$s3. Addi means add immediate, since one of the operand is in immediate addressing mode.
- As per the design principle “Make common case faster”, the constant operands must be loaded faster from the memory.
- Since constants occur more frequently in the instruction, they are mentioned in the instruction itself rather than to load from registers.

Name	Example	Comments
32 Registers	\$S0, \$S1,... \$t0,, \$t1,...	They can be accessed quickly. In MIPS architecture, the data must be loaded into the register to perform arithmetic operation.
2 ³⁰ memory words	Memory[0], Memory[1],...	The contents can be accessed only after data transfer instructions. MIPS use byte addressing.

Category	Instruction	Operation
Arithmetic	Add \$s1, \$s2, \$s3	S1=s2+s3. There are three operands in this instruction. The data resides in the registers.
	Sub \$s1, \$s2, \$s3	S1=s2-s3. There are three operands in this instruction. The data resides in the registers.

	Addi \$s1, \$s2, 50	S1=s2+50. This is add immediate instruction. It has two operands and one constant value, which is directly added to get the result.
Data Transfer	Lw \$s1, 50(\$s2) Sw \$s1, 50(\$s2)	S1=memory [s2+50] Data is transferred from memory to registers. Memory[s2+50]=\$s1 Data is transferred from register to memory.

Representation of Instructions

- Numbers are represented in computer hardware as a series of high and low electronic signals, that are denoted as 0's and 1's. Hence they are considered base 2 numbers.

A bit or binary digit is a single digit of a binary number and is the smallest indivisible unit of computing.

- The binary digit may be used to denote high or low, on or off, true or false, or 1 or 0.
- Registers are part of every instruction, hence there must be a convention to map register names into numbers.

Example: add \$t0,\$s1,\$s2.

This instruction is mapped to its equivalent decimal representation as:

0	17	18	8	0	32
---	----	----	---	---	----

The binary equivalent representation is given as:

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Each cell is termed as a **field**.
- The binary representation used for communication within a computer system is termed as **Machine Language**.
- **Instruction Format** is a representative form an instruction of fields of binary numbers.

Fields in MIPS

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- Opcode is the field that denotes the operation and format of an instruction.

- rs: The first register source operand.
- rt: The second register source operand.
- rd: The register destination operand. It gets the result of the operation.
- shamt: Shift amount. This is done to add two zero's to the low-order end of the sign-extended offset field in calculating the address. This operation truncated the sign values.
- op field and is sometimes called the function code.
- The MIPS instructions are designed in the same format for easy manipulation. This is in accordance with the design principle Good design demands good compromises.

Register Name	Number	Usage
zero	0	Constant 0
at	1	Reserved for assembler
v0	2	Expression evaluation and
v1	3	results of a function
a0	4	Argument 1
a1	5	Argument 2
a2	6	Argument 3
a3	7	Argument 4
t0	8	Temporary (not preserved across call)
t1	9	Temporary (not preserved across call)
t2	10	Temporary (not preserved across call)
t3	11	Temporary (not preserved across call)
t4	12	Temporary (not preserved across call)
t5	13	Temporary (not preserved across call)
t6	14	Temporary (not preserved across call)
t7	15	Temporary (not preserved across call)
s0	16	Saved temporary (preserved across call)
s1	17	Saved temporary (preserved across call)
s2	18	Saved temporary (preserved across call)
s3	19	Saved temporary (preserved across call)
s4	20	Saved temporary (preserved across call)
s5	21	Saved temporary (preserved across call)
s6	22	Saved temporary (preserved across call)
s7	23	Saved temporary (preserved across call)
t8	24	Temporary (not preserved across call)
t9	25	Temporary (not preserved across call)
k0	26	Reserved for OS kernel
k1	27	Reserved for OS kernel
gp	28	Pointer to global area
sp	29	Stack pointer
fp	30	Frame pointer
ra	31	Return address (used by function call)

Fig 1.18: Mapping of register names and numbers

Opcode values of MIPS instruction

In the MIPS instruction reg means a register number ranging from 0 and 31. Address means a 16-bit address, and not applicable (n.a.) means this field does not appear in this format. The add and sub instructions have the same value in the op field. The hardware uses the funct field to decide the whether it is addition or subtraction operation using: add (32) or subtract (34).

Instruction	Format	Op	Rs	Rt	Rd	Shamt	Funct	Address
Add	R	0	Reg	Reg	Reg	0	32_{10}	Na
Sub	R	0	Reg	Reg	Reg	0	34_{10}	Na
Add immediate	I	8_{10}	Reg	Reg	Na	Na	Na	Constant
Lw	I	35_{10}	Reg	Reg	Na	Na	Na	Address
Sw	I	43_{10}	Reg	Reg	Na	Na	Na	Address

Logical Operations

The following are the logical operations performed by the processor:

Logical Operations	MIPS Instructions
Shift left	sll
Shift right	srl
Bit by bit AND	and, andi
Bit by bit OR	or, ori
Bit by bit NOT	nor

The first class of such operations is called **shifts**. They move all the bits in a word to the left or right, filling the emptied bits with 0s.

$$0000\ 0000\ 0000\ 00000\ 000\ 0000\ 0000\ 0000\ 1001_2 = 9_{10}$$

After left shifting by four, the new value is 144.

$$0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1001\ 0000_2 = 144_{10}$$

- **Left shift:** Left shifting by i bits is equivalent to multiplying the number by 2^i .
- **Right Shift:** Right shifting by i bits is equivalent to dividing the number by 2^i .
- **AND:** This is used in masking of bits.
- **OR:** It is a bit-by-bit operation that places a 1 in the result if either operand bit is a 1
- **NOT:** A logical bit-by-bit operation with one operand that inverts the bits; that is, it replaces every 1 with a 0, and every 0 with a 1.
- **NOR:** A logical bit-by-bit operation with two operands that calculates the NOT of the OR of the two operands.

Category	Instruction	Operation
AND	and \$s1, \$s2, \$s3	$S1=s2\&s3$
OR	or \$s1, \$s2, \$s3	$S1=s2 s3$
NOR	nor \$s1, \$s2, \$s3	$S1=\sim(s2 s3)$
NAND	nand \$s1, \$s2, \$s3	$S1=\sim(s2\&s3)$
AND immediate	andi \$s1, \$s2, 100	$S1=s2\&100$
OR immediate	ori \$s1, \$s2, \$s3	$S1=s2 100$
Shift left logical	sll \$s1, \$s2, 10	$S1=s2\ll 10$
Shift right logical	srl \$s1, \$s2, 10	$S1=s2\gg 10$

Control Operations

Decision making and branching makes the computers more powerful.

Decision Making:

Decision making in MIPS assembly language includes two decision-making instructions (conditional branches):

i) Branch if Equal (BEQ):

```
beq register1, register2, L1
```

In this instruction, the go to the statement labeled L1 if the value in register1 is equal to the value in register2.

ii) Branch if not Equal (BNE):

```
bne register1, register2, L1
```

In this instruction, the go to the statement labeled L1 if the value in register1 does not equal the value in register2.

Conditional branch is an instruction that requires the comparison of two values and that allows for a subsequent transfer of control to a new address in the program based on the outcome of the comparison.

Example:

Consider the following statement,

```
if (i == j) f = g + h; else f = g - h;
```

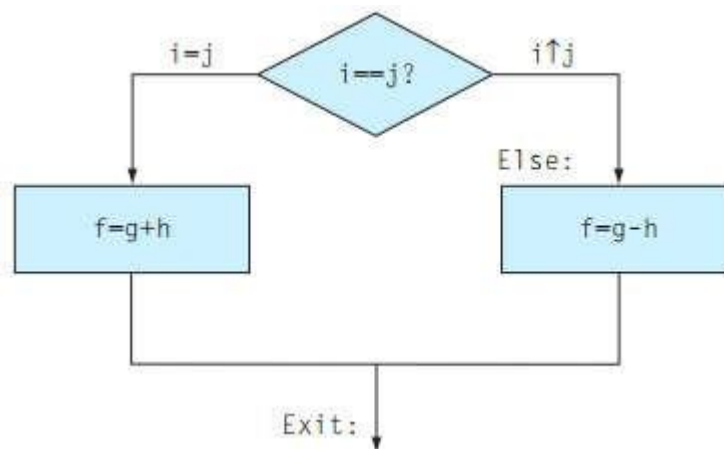


Fig 1.19: Flowchart for if (i == j) f = g + h; else f = g - h;

The instruction first compares for equality, using beq. In general, the code will be more efficient if we test for the opposite condition to branch over the code that performs the subsequent then part of the if (the label Else is defined below):

```
bne $s3,$s4,Else # go to Else if i ≠ j
```

The next assignment statement performs a single operation, and if all the operands are allocated to registers, it is just one instruction:

```
add $s0,$s1,$s2 # f = g + h (skipped if i ≠ j)
```

This instruction says that the processor always follows the branch. To distinguish between conditional and unconditional branches, the MIPS name for this type of instruction is `jump`, abbreviated as `j` (the label `Exit` is defined below).

```
j Exit # go to Exit
```

The assignment statement in the else portion of the if statement can again be compiled into a single instruction. We just need to append the label `Else` to this instruction. We also show the label `Exit` that is after this instruction, showing the end of the if-then-else compiled code:

```
Else:sub $s0,$s1,$s2 # f = g - h (skipped if i = j)
```

```
Exit:
```

Compilers create branches and labels wherever necessary for maintaining flow of the program. Also, the assembler calculates the addresses and relieves the compiler and the assembly language programmer.

Looping:

When a set of statements has to be executed more number of times, looping statements are used.

Example:

```
while (save[i] == k)
```

```
    i += 1;
```

`i` and `k` correspond to registers `$s3` and `$s5` and the base of the array `save` is in `$s6`. The MIPS instructions are:

- The first step is to load `save[i]` into a temporary register. This operation needs an address. Multiply the index `i` by 4 and add `i` to the base of array to obtain the address.
- Add the label `Loop` to it to branch back to that instruction at the end of the loop:

```
Loop: sll $t1,$s3,2 # Temp reg $t1 = 4 * i
```

- To get the address of `save[i]`, add `$t1` and the base of `save` in `$s6`:

```
add $t1,$t1,$s6 # $t1 = address of save[i]
```

- Use that address to load `save[i]` into a temporary register:

```
lw $t0,0($t1) # Temp reg $t0 = save[i]
```

- The next instruction performs the loop test, exiting if save[i] ≠k:
bne \$t0,\$s5, Exit # go to Exit if save[i] ≠ k
- The next instruction adds 1 to i :
add \$s3,\$s3,1 # i = i + 1
- The end of the loop branches back to the while test at the top of the loop. Add the Exit label after it:
j Loop # go to Loop

Exit:

Grouping on instructions that makes compiling easy is through partitioning the assembly language instructions into basic blocks.

A sequence of instructions without branches except possibly at the end and without branch targets or branch labels except possibly at the beginning are called basic blocks.

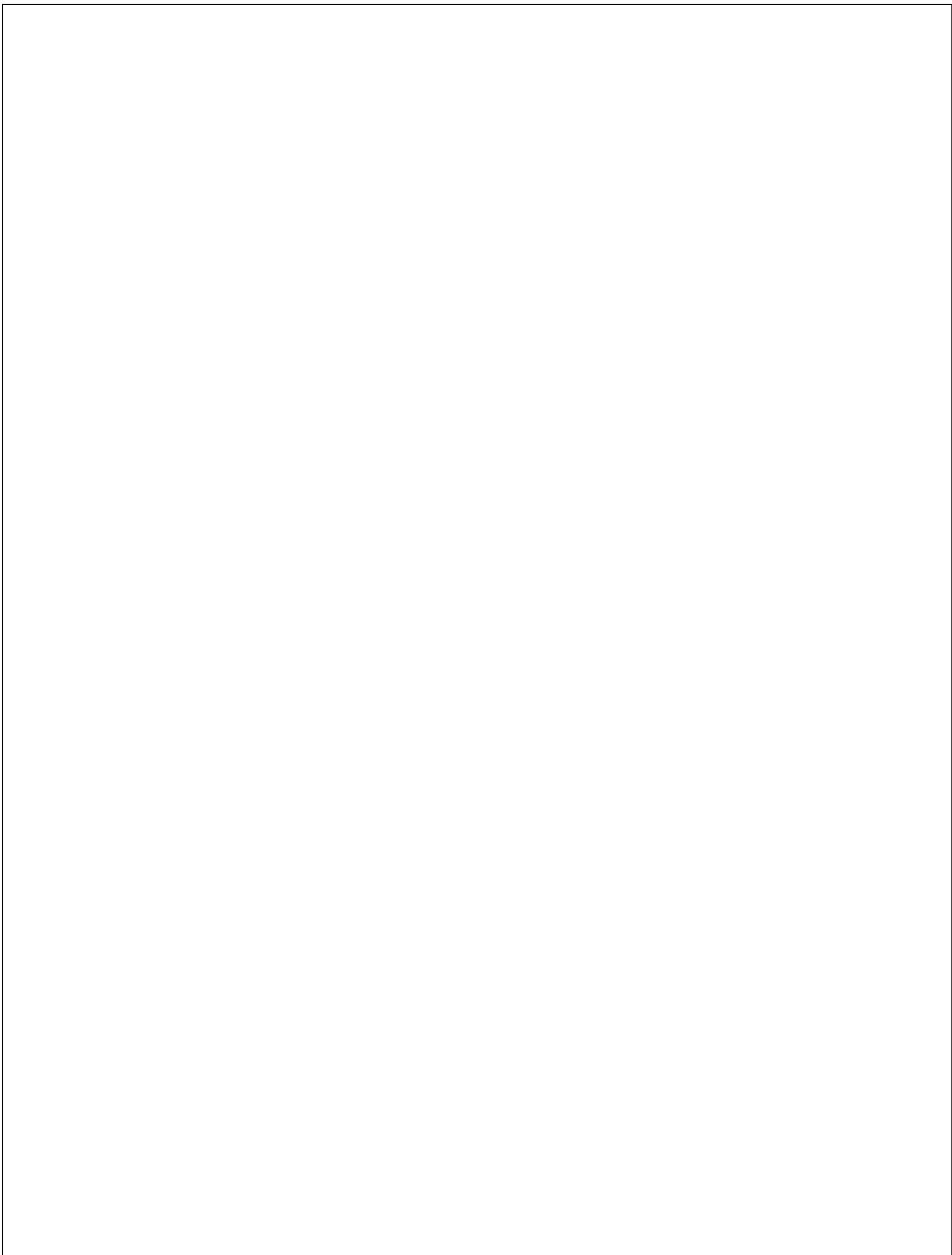
Case / Switch Statements

These statements allow the programmers to select one among the many options. The simple way to implement switch is through a sequence of conditional tests using a chain of if-then-else statements. The alternatives are encoded in **jump address table**. The program needs only to index into the table and then jump to the appropriate sequence.

A table of addresses of alternative instruction sequences is maintained in jump address table.

The jump table is an array of words containing addresses that correspond to labels in the code. MIPS include a **jump register instruction** (jr), to support the unconditional jump to the address specified in a register. The program loads the appropriate entry from the jump table into a register, and then it jumps to the proper address using a jump register.

Category	Instruction	Operation
Conditional Branch	Beq \$s1, \$s2,L	If (s1==s2) then goto L (Branch if equal)
	Bne \$s1, \$s2,L	If (s1≠s2) then goto L (Branch if not equal)
	Slt \$s1, \$s3, \$s3	If (s2<s3) set s1=1Else s1=0(set on less than)
	Slt\$s1, \$s2, 100	If (s2<100) set s1=1Else s1=0(set on less than immediate)
Unconditional branch	J L	Goto L (Jump to target address L)



UNIT - II

ARITHMETIC

INTRODUCTION

Data is manipulated by using the arithmetic instructions in digital computers to give solution for the computation problems. The addition, subtraction, multiplication and division are the four basic arithmetic operations. **Arithmetic processing unit** is responsible for executing these operations and it is located in central processing unit.

The arithmetic instructions are performed on binary or decimal data. **Fixed-point numbers** are used to represent integers or fractions. These numbers can be signed or unsigned negative numbers. A wide range of arithmetic operations can be derived from the basic operations.

Signed and Unsigned Numbers:

Signed numbers:

These numbers require an arithmetic sign. The most significant bit of a binary number is used to represent the sign bit. If the sign bit is equal to zero, the signed binary number is positive; otherwise, it is negative. The remaining bits represent the actual number. The negative numbers may be represented either in a signed magnitude or signed complement representation. There are three ways of representing negative fixed point

- Binary numbers signed magnitude
- Signed 1's complement
- Signed 2's complement

Unsigned binary numbers:

These are positive numbers and thus do not require an arithmetic sign. An m -bit unsigned number represents all numbers in the range 0 to $2^m - 1$. For example, the range of 16-bit unsigned binary numbers is from 0 to $65,535$ in decimal and from 0000 to $FFFF$ in hexadecimal.

Signed Magnitude Representation:

The most significant bit (MSB) represents the *sign*. A 1 in the MSB bit position denotes a negative number and 0 denotes a positive number. The remaining $n - 1$ bits are preserved and represent the magnitude of the number.

Examples:

Number	Signed Magnitude Representation
+3	0011
-3	1011
0	0000
-0	1011
5	0101
-5	1101

One's Complement Representation:

In one's complement, positive numbers remain unchanged as before with the sign-magnitude numbers. Negative numbers are represented by taking the one's complement (inversion, negation) of the unsigned positive number. Since positive numbers always start with a 0, the complement will always start with a 1 to indicate a negative number.

The one's complement of a negative binary number is the complement of its positive counterpart, so to take the one's complement of a binary number.

Number	One's complement Representation
00001000 (+8)	11110111
10001000(-8)	01110111
00001100(+12)	11110011
10001100(-12)	01110011

Two's Complement Representation:

In two's complement, the positive numbers are exactly the same as before for unsigned binary numbers. A negative number, is represented by a binary number, which when added to its corresponding positive equivalent results in zero.

In two's complement form, a negative number is the 2's complement of its positive number with the subtraction of two numbers being $A - B = A + (\text{2's complement of } B)$ using much the same process as before as basically, two's complement is adding 1 to one's complement of the number.

The main difference between 1's complement and 2's complement is that 1's complement has two representations of 0 (+0): 00000000, and (-0): 11111111. In 2's complement, there is only one representation for zero: 00000000 (0).

+0: 00000000

2's complement of -0:

-0: 00000000 (Signed magnitude representation)

11111111 (1's complement representation)

11111111 + 1 = 00000000 (2's complement representation)

These shows in 2's complement representation both +0 and -0 takes same value. This solves the **double-zero problem**, which existed in the 1's complement.

Example 2.1: Convert 2_{10} and -2_{10} to 32 bit binary numbers.

+2 = 0000 0000 0000 0010 (16 bits)

= 0000 0000 0000 0000 0000 0000 0000 0010 (32 bits)

It is converted to a 32-bit number by making 16 copies of the value in the most significant bit (0) and placing that in the left-hand half of the word.

2 = 0000 0000 0000 0010

-2 = 1's complement of 2 + 1

1111 1111 1111 1101 (1's complement of 2) + 1

= 1111 1111 1111 1110 (16 bits)

= 1111 1111 1111 1111 1111 1111 1111 1110 (32 bits)

To convert to 32 bit number copy the digit in the MSB of the 16 bit number for 16 times and fill the left half.

FIXED POINT ARITHMETIC

A fixed-point number representation is a real data type for a number that has a fixed number of digits after the radix point or decimal point.

This is a common method of integer representation is sign and magnitude representation. One bit is used for denoting the sign and the remaining bits denote the magnitude. With 7 bits reserved for the magnitude, the largest and smallest numbers represented are +127 and -127. Fixed-point numbers are useful for representing fractional values, usually in base 2 or base 10, when the executing processor has no floating point unit (FPU) or if fixed-point provides improved performance or accuracy for the application at hand. Most low-cost embedded microprocessors and microcontrollers do not have an FPU.

A value of a fixed-point data type is essentially an integer that is scaled by a specific factor. The scaling factor is usually a power of 10 (for human convenience) or a power of 2 (for computational efficiency). However, other scaling factors may be used occasionally, e.g. a time value in hours may be represented as a fixed-point type with a scale factor of 1/3600 to obtain values with one-second accuracy. The maximum value of a fixed-point type is the largest value that can be represented in the underlying integer type, multiplied by the scaling factor; and similarly for the minimum value.

Example:

The value 1.23 can be represented as 1230 in a fixed-point data type with scaling factor of 1/1000.

Precision loss and overflow

- The fixed point operations can produce results that have more bits than the operands there is possibility for information loss.
- In order to fit the result into the same number of bits as the operands, the answer must be rounded or truncated.
- Fractional bits lost below this value represent a precision loss which is common in fractional multiplication.
- If any integer bits are lost, however, the value will be radically inaccurate.
- Some operations, like divide, often have built-in result limiting so that any positive overflow results in the largest possible number that can be represented by the current format.

- Likewise, negative overflow results in the largest negative number represented by the current format. This built in limiting is often referred to as **saturation**.
- Some processors support a hardware overflow flag that can generate an exception on the occurrence of an overflow, but it is usually too late to salvage the proper result at this point.

Addition and Subtraction

In addition, the digits are added bit by bit from right to left, with carries passed to the next digit to the left. Subtraction operation is also done using addition: The appropriate operand is simply negated before being added.

Addition: $A + B$; A: Augend; B: Addend
Subtraction: $A - B$: A: Minuend; B: Subtrahend

Operation	Add Magnitude	Subtract Magnitude		
		When $A > B$	When $A < B$	When $A = B$
$(+A) + (+B)$	$+(A + B)$			
$(+A) + (-B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(-A) + (+B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$
$(-A) + (-B)$	$-(A + B)$			
$(+A) - (+B)$		$+(A - B)$	$-(B - A)$	$+(A - B)$
$(+A) - (-B)$	$+(A + B)$			
$(-A) - (+B)$	$-(A + B)$			
$(-A) - (-B)$		$-(A - B)$	$+(B - A)$	$+(A - B)$

Fig 2.1: Addition and Subtraction operation

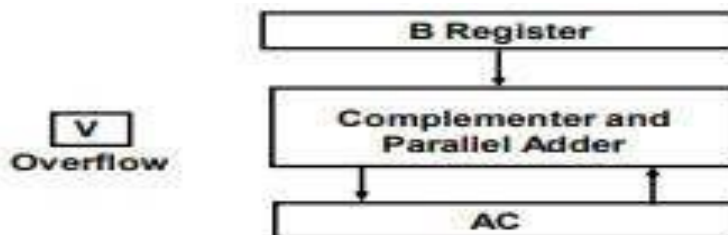


Fig 2.2: Hardware for addition / subtraction

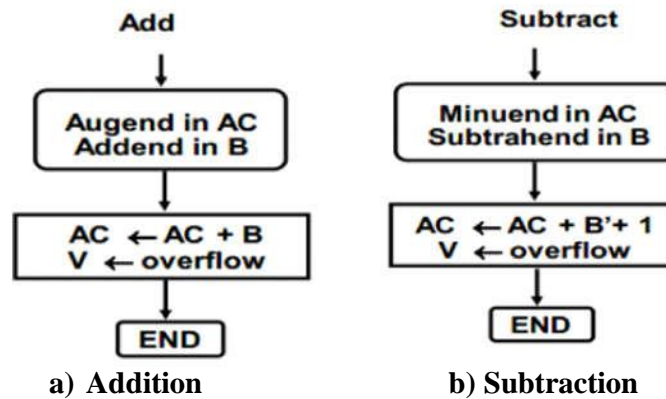


Fig 2.2: Addition and subtraction algorithm

Steps for addition:

- Place the addend in register B and augend in AC.
- Add the contents in B and AC and place the result in AC.
- V register will hold the overflow bits (if any).

Steps for subtraction:

- Place the minuend in AC and subtrahend in B.
- Add the contents of AC and 2's complemented B. Place the result in AC.
- V register will hold the overflow bits (if any).

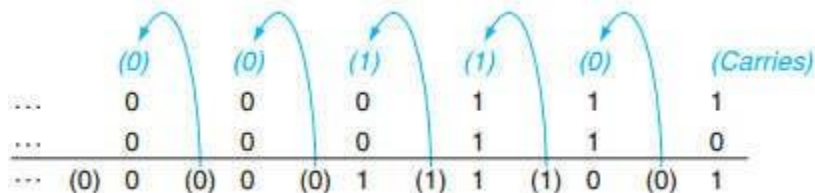


Fig 2.3: Manipulating carry

The figure 2.3 shows binary addition with carries from right to left. The rightmost bit adds 1 to 0, resulting in the sum of this bit being 1 and the carry out from this bit being 0. Hence, the operation for the second digit to the right is $0 + 1 + 1$. This generates a 0 for this sum bit and a carry out of 1. The third digit is the sum of $1 + 1 + 1$, resulting in a carry out of 1 and a sum bit of 1. The fourth bit is $1 + 0 + 0$, yielding a 1 sum and no carry. If there is a carry at this bit, it will be stored in the overflow register.

Overflow occurs in subtraction when we subtract a negative number from a positive number and get a negative result, or when we subtract a positive number from a negative number and get a positive result. This means a borrow occurred from the sign bit.

Operation	Operand A	Operand B	Result indicating overflow
A+B	≥ 0	≥ 0	< 0
A+B	< 0	< 0	≥ 0
A-B	≥ 0	< 0	< 0
A-B	< 0	≥ 0	≥ 0

Example 2.2: Add 6 and 7.

$$\begin{array}{r}
 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\
 +\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\
 \hline
 =\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101_{two} = 13_{ten}
 \end{array}$$

Example 2.3: Subtract 6 from 7.

$$\begin{array}{r}
 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\
 -\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0110_{two} = 6_{ten} \\
 \hline
 =\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}
 \end{array}$$

Example 2.4: Subtract 6 from 7 through 2's complement.

$$\begin{array}{r}
 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0111_{two} = 7_{ten} \\
 +\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1010_{two} = -6_{ten} \\
 \hline
 =\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0001_{two} = 1_{ten}
 \end{array}$$

The MIPS instructions for addition and subtraction are given in the following table:

Instruction	Example	Operation
Add	Add \$s1, \$s2, \$s3	$S1=s2+s3$ Overflow detected
Subtract	Sub \$s1, \$s2, \$s3	$S1=s2-s3$ Overflow detected
Add Immediate	Addi \$s1, \$s2, 100	$S1=s2+100$ Overflow detected
Add unsigned	Addu \$s1, \$s2, \$s3	$S1=s2+s3$ Overflow undetected
Subtract unsigned	Subu \$s1, \$s2, \$s3	$S1=s2-s3$ Overflow undetected
Add immediate unsigned	Addiu \$s1, \$s2, 100	$S1=s2+100$ Overflow undetected

Multiplication

Multiplication is seen as repeated addition. The first operand is called the multiplicand and the second the multiplier. The final result is called the product. The number of digits in the product is larger than the number in either the multiplicand or the multiplier. The length of the multiplication of an n -bit multiplicand and an m -bit multiplier is a product that is $n + m$ bits long. The steps in multiplication are:

1. Place a copy of the in the proper place if the multiplier digit is a 1
2. Place 0 in the proper place if the digit is 0.

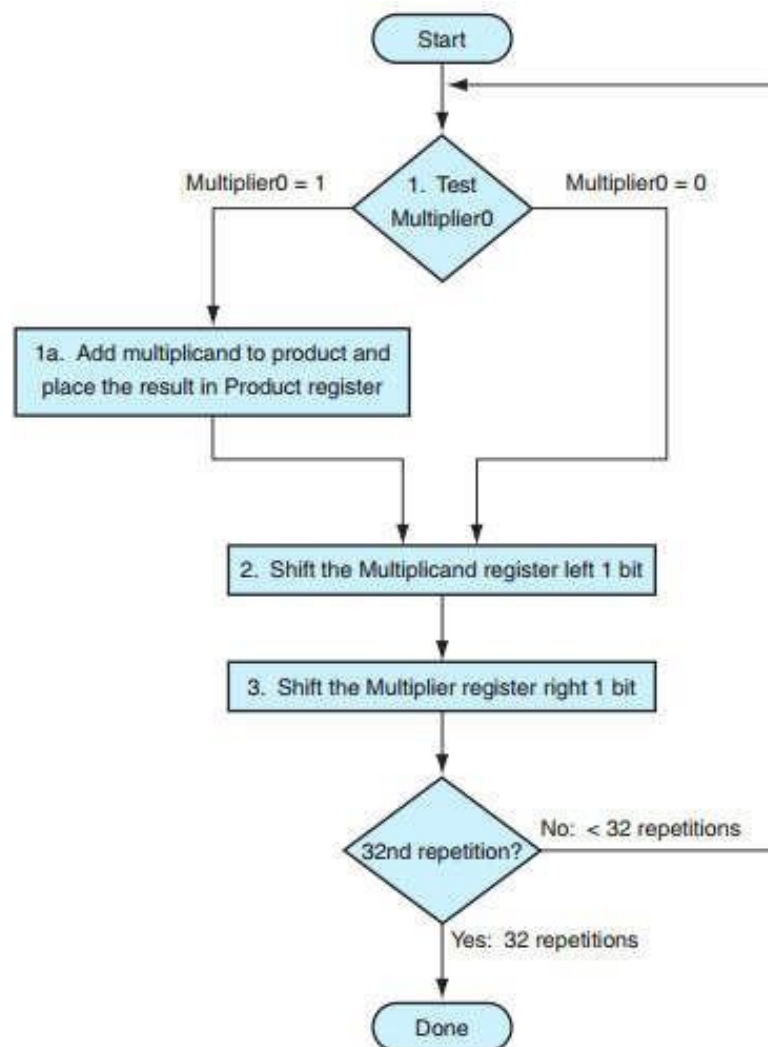


Fig 2.4: Basic multiplication algorithm

Booth's Algorithm:

Booth algorithm gives a procedure for multiplying binary integers in signed-2's complement representation. It operates on the fact that strings of 0's in the multiplier require no addition but just shifting, and a string of 1's in the multiplier from bit weight 2^k to weight 2^m can be treated as $2^{k+1} - 2^m$.

For example, the binary number 001110 (+14) has a string 1's from 23 to 21 ($k=3$, $m=1$). The number can be represented as $2^{k+1} - 2^m = 2^4 - 2^1 = 16 - 2 = 14$. Therefore, the multiplication $M \times 14$, where M is the multiplicand and 14 the multiplier, can be done as $M \times 2^4 - M \times 2^1$. Thus the product can be obtained by shifting the binary multiplicand M four times to the left and subtracting M shifted left once.

Booth algorithm requires examination of the multiplier bits and shifting of partial product. Prior to the shifting, the multiplicand may be added to the partial product, subtracted from the partial, or left unchanged according to the following rules:

1. The multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of 1's in the multiplier.
2. The multiplicand is added to the partial product upon encountering the first 0 in a string of 0's in the multiplier.
3. The partial product does not change when multiplier bit is identical to the previous multiplier bit.

The algorithm works for positive or negative multipliers in 2's complement representation. This is because a negative multiplier ends with a string of 1's and the last operation will be a subtraction of the appropriate weight. The two bits of the multiplier in Q_n and Q_{n+1} are inspected. If the two bits are equal to 10, it means that the first 1 in a string of 1's has been encountered. This requires a subtraction of the multiplicand from the partial product in AC. If the two bits are equal to 01, it means that the first 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the partial product in AC. When the two bits are equal, the partial product does not change.

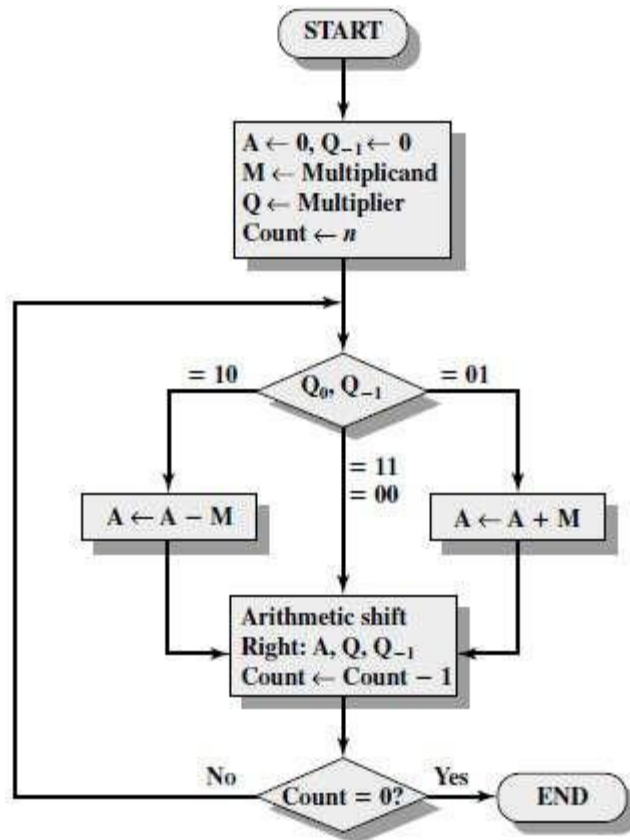


Fig 2.5: Flowchart for Booth's algorithm

Example 2.5: Multiply 7 and 3 using Booth's algorithm.

A	Q	Q ₋₁	M		
0000	0011	0	0111	Initial values	
1001	0011	0	0111	A ← A - M	} First cycle
1100	1001	1	0111	Shift	
1110	0100	1	0111	Shift	} Second cycle
0101	0100	1	0111	A ← A + M	} Third cycle
0010	1010	0	0111	Shift	
0001	0101	0	0111	Shift	} Fourth cycle

The product is available in AQ.

Example 2.6 : Multiply -5 and -7 using Booth's algorithm

A	Q	Q-1	M
0000	1001	0	4
0101	1001	0	
0010	1100	1	3
1101	1100	1	
1110	1110	0	2
1111	0111	0	1
0010	0011	1	0

The product is available in AQ

Division

Division is repeated subtraction. The two operands (dividend and divisor) and the result (quotient) of divide are accompanied by a second result called the remainder. The following are the terminologies:

- Dividend: A number being divided.
- Divisor: A number that the dividend is divided by.
- Quotient: The primary result of a division; a number that when multiplied by the divisor and added to the remainder produces the dividend.
- Remainder: The secondary result of a division; a number that when added to the product of the quotient and the divisor produces the dividend

$$\text{Dividend} = \text{Quotient} * \text{Divisor} + \text{Remainder}$$

Divisor	1000_{ten}	$\overline{)1001010_{\text{ten}}}$	1001_{ten}	Quotient
		$\underline{-1000}$		Dividend
		10		
		101		
		1010		
		$\underline{-1000}$		
		10_{ten}		Remainder

Fig 2.6: Division Terminologies

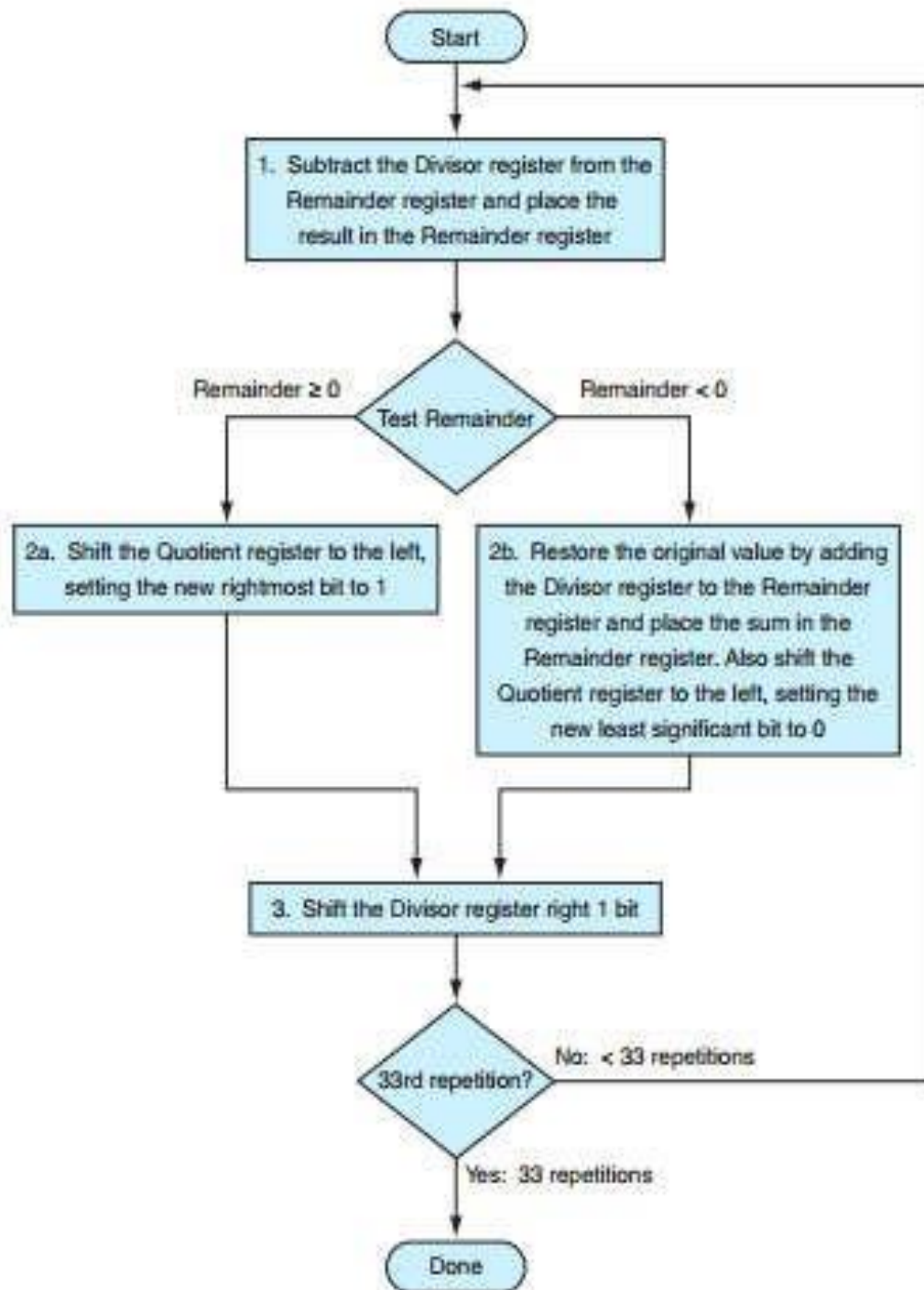


Fig 2.7: Basic division operation

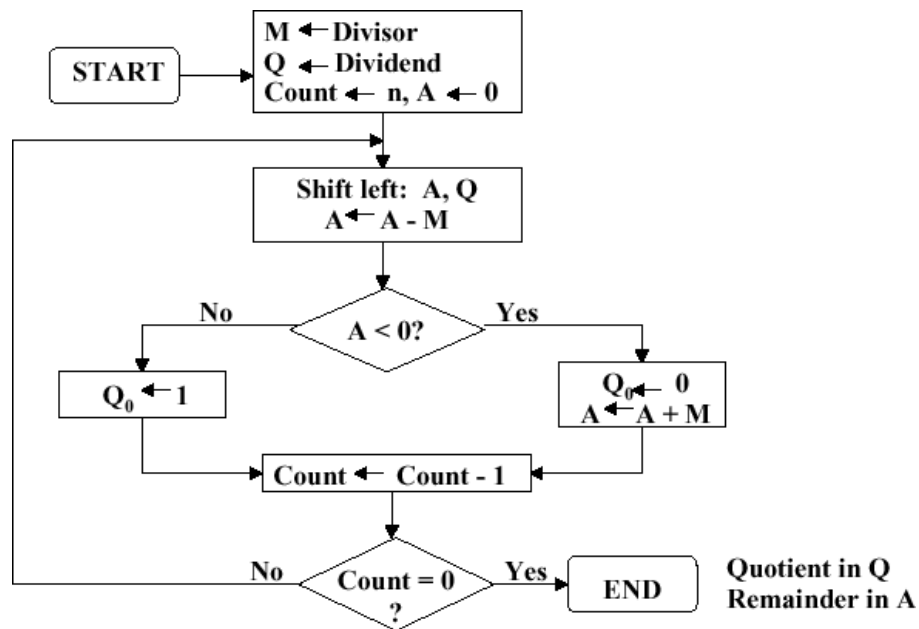


Fig 2.8: Fixed point division

Example 2.7: Divide -7 by 3

A	Q	M = 0011	
0000	0111		Initial values
0000	1110		Shift
1101		A = A - M	} 1
0000	1110	A = A + M	
0001	1100		Shift
1110		A = A - M	} 2
0001	1100	A = A + M	
0011	1000		Shift
0000		A = A - M	} 3
0000	1001	Q₀ = 1	
0001	0010		Shift
1110		A = A - M	} 4
0001	0010	A = A + M	

Quotient=0010Remainder=0001

Example 2.8: Divide -7 by -3

A	Q	M = 1101
1111	1001	Initial values
1111	0010	Shift
0010		Subtract
1111	0010	Restore
1110	0100	Shift
0001		Subtract
1110	0100	Restore
1100	1000	Shift
1111		Subtract
1111	1001	$Q_0 = 1$
1111	0010	Shift
0010		Subtract
1111	0010	Restore

Example 2.9: Divide 7 by 3

A	Q	M = 0011
0000	0111	Initial values
0000	1110	Shift
1101		Subtract
0000	1110	Restore
0001	1100	Shift
1110		Subtract
0001	1100	Restore
0011	1000	Shift
0000		Subtract
0000	1001	$Q_0 = 1$
0001	0010	Shift
1110		Subtract
0001	0010	Restore

Example 2.10: Divide -7 by 3

A	Q	M = 0011	
1111	1001		Initial values
1111	0010		Shift
0010			Add
1111	0010		Restore
			} 1
1110	0100		Shift
0001			Add
1110	0100		Restore
			} 2
1100	1000		Shift
1111			Add
1111	1001		Q ₀ = 1
1111	0010		Shift
0010			Add
1111	0010		Restore
			} 4

MIPS instructions for multiplication and division

Category	Example	Description
Multiply	mult \$s2, \$s3	Hi, lo=s2 * s3 64 bit signed product in Hi, Lo
Multiply unsigned	multu \$s2, \$s3	Hi, lo=s2 * s3 64 bit signed product in Hi, Lo
Divide	div \$s2, \$s3	Lo=s2/s3 (Quotient) Hi=s2 mod s3 (Remainder)
Divide unsigned	divu \$s2, \$s3	Lo=s2/s3 (unsigned Quotient) Hi=s2 mod s3 (Remainder)
Move from Hi	mfhi \$s1	S1=Hi Used to get a copy of Hi
Move from Lo	mflo \$s1	S1=lo Used to get a copy of Lo

FLOATING POINT ARITHMETIC

To represent the fractional binary numbers (IEEE 754 floating point format), it is necessary to consider floating point. If the point is assumed to the right of the sign bit, we can represent the fractional binary numbers as given below:

$$B = (b_0 * 2^0 + b_1 * 2^{-1} + b_2 * 2^{-2} + \dots + b_{(n-1)} * 2^{-(n-1)})$$

With this fractional number system, we can represent the fractional numbers in the following range,

$$-1 < F < 1 - 2^{-(n-1)}$$

The binary point is said to be float and the numbers are called **floating point numbers**. The position of binary point in floating point numbers is variable and hence numbers must be represented in the specific manner is referred to as floating point representation. The floating point representation has three fields. They are:

- **Sign:** Sign bit is the first bit of the binary representation. '1' implies negative number and '0' implies positive number.

Example: 11000001110100000000000000000001. This is negative number since it starts with 1.

- **Exponent:** It starts from bit next to the sign bit of the binary representation. The exponent field is needed to represent both positive and negative exponents. To do this, a bias is added to the actual exponent in order to get the stored exponent. For IEEE single-precision floats, this value is 127. Thus, to express an exponent of zero, 127 is stored in the exponent field. A stored value of 200 indicates an exponent of (200-127), or 73. The exponents of "127 (all 0s) and +128 (all 1s) are reserved for special numbers.

Double precision has an 11-bit exponent field, with a bias of 1023. **Example:** For 8 bit conversion: $8 = 2^{3-1} - 1 = 3$. Bias=3.

For 32 bit conversion: $32 = 2^{8-1} - 1 = 127$. Bias=127.

- **Significant digits or Mantissa:** It is calculated from the remaining 23 bits of the binary representation. It consists of '1' and a fractional part. This represents the

precision bits of the number. It is composed of an implicit leading bit (left of the radix point) and the fraction bits (to the right of the radix point). To find out the value of the implicit leading bit, consider that any number can be expressed in scientific notation in many different ways.

Example: 50 can be represented as

1. 0.050×10^3
2. $.5000 \times 10^3$
3. 5.000×10^1
4. 50.00×10^0
5. $5000. \times 10^{-2}$

In order to maximize the quantity of representable numbers, floating-point numbers are typically stored in normalized form. This basically puts the radix point after the first non-zero digit. In normalized form, 50 is represented as 5.000×10^1 .

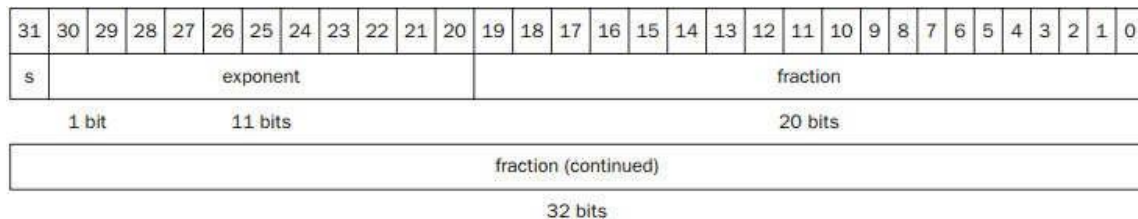


Fig 2.9: Parts of floating point number

Conversion of Decimal number to floating point:

- **Sign bit:** 1 implies negative number and 0 implies positive number.
- **Exponent:** To find the exponent value for binary representation, express the number by the nearest smaller or equal to 2^k number. The bias is determined by $2^{k-1}-1$, where 'k' is the number of bits in exponent field. Add the bias with k value to express the exponent in binary form.
- **Mantissa:** Move the binary point so that there is only one bit from the left. Adjust the exponent of 2 so that the value does not change. This is normalizing the number. Now, consider the fractional part and represented as 23 bits by adding zeros.

Example 2.11. Find the decimal equivalent of the floating point number:

01000001110100000000000000000000

Sign=0

Exponent:

10000011=131₁₀

131-127=4

Exponent= 2⁴=16

Mantissa:

Remaining 23 bits: 10100000000000000000000

=1*(1/2) + 0*(1/4) + 1*(1/8) + 0*(1/16) +..... = 0.625

Decimal number= Sign * Exponent * Mantissa

=-1 * 16 *0.625

= -26

Example 2.11: Find the floating point equivalent of -17.

Sign=1 (-ve number)

Exponent:

Bias for 32 bit = 127 (2⁸⁻¹ -1 = 127)

127 + 4 = 131=10000011₂

Mantissa:

17 = 10001₂=1.0001 x 2⁴

Fractional part=000100000000000000000000

-17 =1 10000011 000100000000000000000000₂

Terminologies:

- **Overflow:** A situation in which a positive exponent becomes too large to fit in the exponent field.
- **Underflow:** A situation in which a negative exponent becomes too large to fit in the exponent field.
- **Double precision:** A floating point value represented in two 32-bit words.

- **Single precision:** A floating point value represented in a single 32-bit word.

	Sign	Exponent	Fraction
Single Precision	1 [31]	8 [30–23]	23 [22–00]
Double Precision	1 [63]	11 [62–52]	52 [51–00]

Fig 2.10: Floating point formats

Example 2.12: The IEEE-754 32-bit floating-point representation pattern is 0 10000000 110 0000 0000 0000 0000 0000. What is the number?

Sign bit $S = 0$ (positive number)

Exponent $E = 10000000_2 = 128_{10}$ (in normalized form)

Fraction is 1.11_2 (with an implicit leading 1) $= 1 + 1 \times 2^{-1} + 1 \times 2^{-2} = 1.75_{10}$

The number is $+1.75 \times 2^{(128-127)} = +3.5_{10}$

Example 2.13: Suppose that IEEE-754 32-bit floating-point representation pattern is 1 01111110 100 0000 0000 0000 0000 0000. Find the decimal number.

Sign bit $S = 1$ (negative number)

$E = 0111\ 1110_2 = 126_{10}$ (in normalized form)

Fraction is 1.1_2 (with an implicit leading 1) $= 1 + 2^{-1} = 1.5_{10}$

The number is $-1.5 \times 2^{(126-127)} = -0.75_D$

Example 2.14: Suppose that IEEE-754 32-bit floating-point representation pattern is 1 01111110 000 0000 0000 0000 0000 0001. What is the decimal number?

Sign bit $S = 1$ (negative number) $E = 0111\ 1110_2 = 126_{10}$ (in normalized form)

Fraction is $1.000\ 0000\ 0000\ 0000\ 0000\ 0001_B$ (with an implicit leading 1) $= 1 + 2^{-23}$

The number is $-(1 + 2^{-23}) \times 2^{(126-127)} = -0.500000059604644775390625$

Example 2.15: Express 85.125 in single and double precision.

$85 = 1010101$

$0.125 = 001$

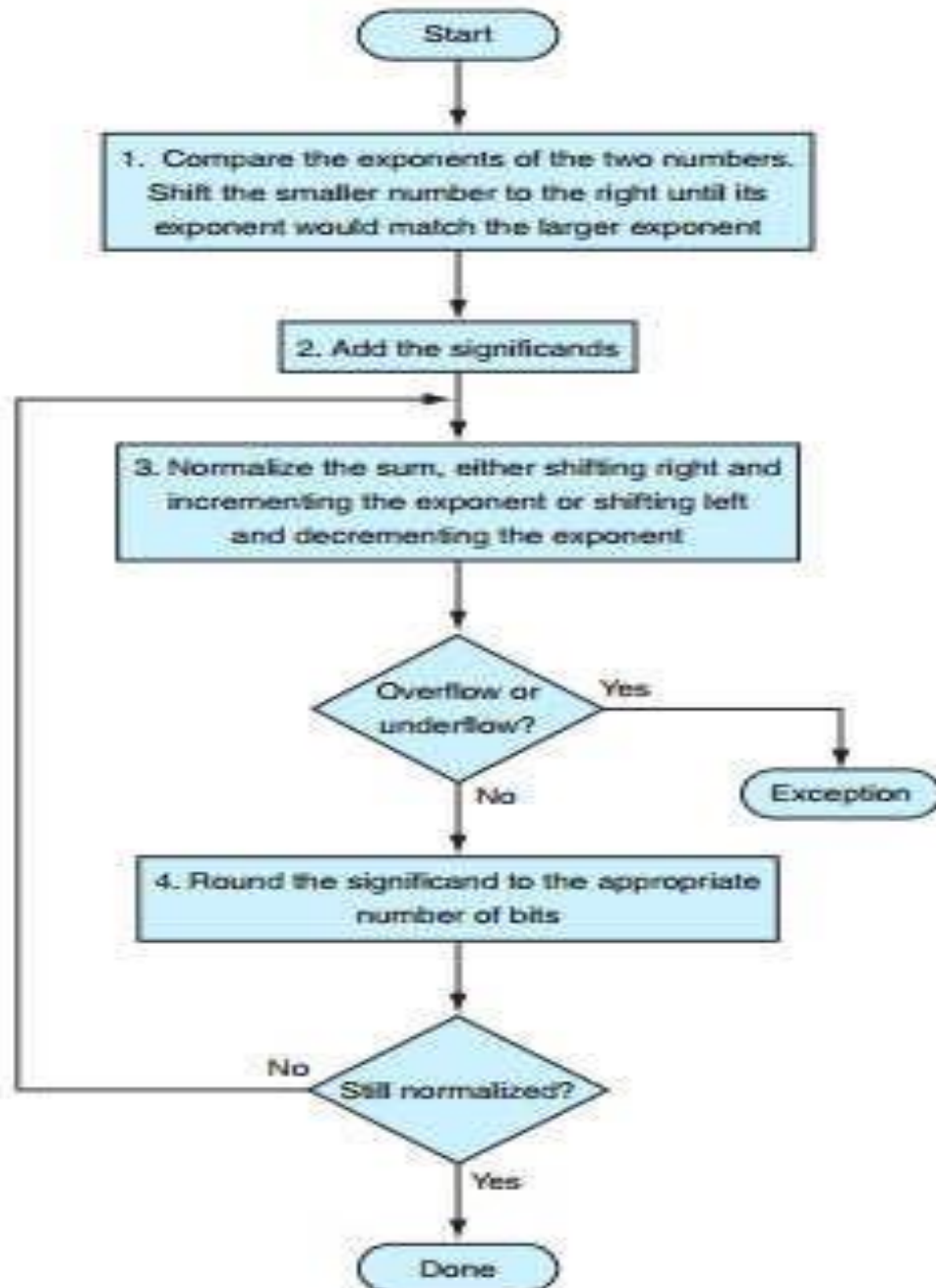


Fig 2.11: Flowchart for floating point addition / subtraction

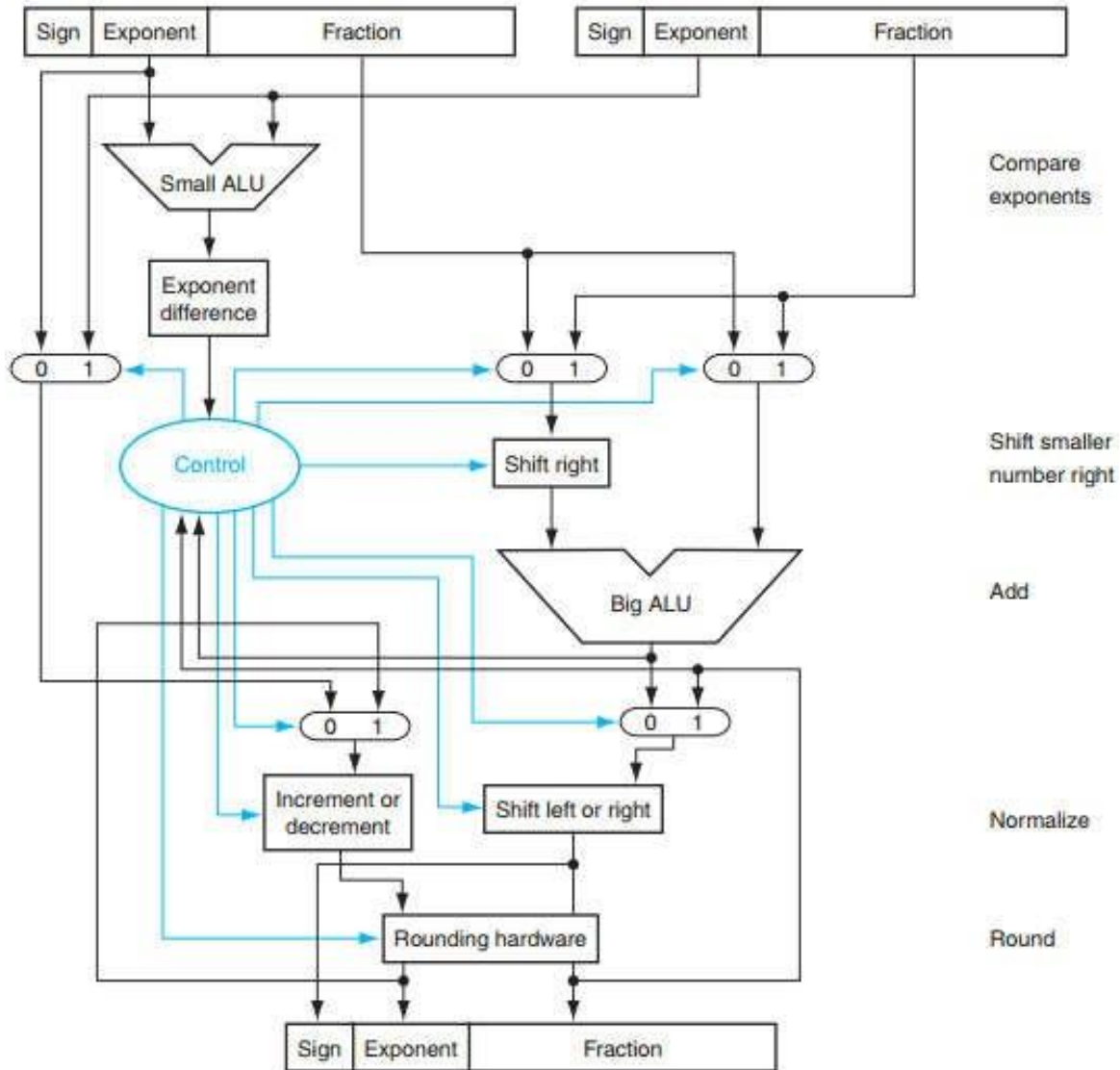


Fig 2.12: Hardware for floating point addition

The addition operation proceeds as the exponent of one operand is subtracted from the other using the small ALU to determine which is larger and by how much. This difference controls the three multiplexors; from left to right, they select the larger exponent, the significand of the smaller number, and the significand of the larger number. The smaller significand is shifted right, and then the significands are added together using the big ALU.

The normalization step then shifts the sum left or right and increments or decrements the exponent. Rounding then creates the final result, which may require normalizing again to produce the final result.

Example 2.16: Add $0.5 + (-0.4375)$

$$0.5 = 0.1 \times 2^0 = 1.000 \times 2^{-1} \text{ (normalised)}$$

$$-0.4375 = -0.0111 \times 2^0 = -1.110 \times 2^{-2} \text{ (normalised)}$$

Step 1: Rewrite the smaller number such that its exponent matches with the exponent of the larger number.

$$-1.110 \times 2^{-2} = -0.1110 \times 2^{-1}$$

Step 2: Add the mantissas

$$\begin{array}{r} 1.000 \times 2^{-1} + \\ -0.1110 \times 2^{-1} \\ \hline 0.001 \times 2^{-1} \end{array}$$

Step 3: Renormalize the mantissa by shifting mantissa and adjusting the exponent. $0.001 \times 2^{-1} = 1.000 \times 2^{-4}$

$-126 \leq -4 \leq 127$ (-4 is within the range of -126 and 127). No overflow or underflow

Step 4: The sum fits in 4 bits so rounding is not required

Example 2.17: Express the following numbers in IEEE 754 format and find their sum:

2345.125 and 0.75. Single precision format of 2345.125:

0	10001010	001001010010010000000000
---	----------	--------------------------

Single precision format of 0.75:

0	01111110	100000000000000000000000
---	----------	--------------------------

Exponent of 2345.125 > exponent of 0.75

$$10001010 - 01111110 = 00000110 = (12)_{10}$$

Shift 0.75 to 12 positions right: 0.000000000001100000000000

Add:

$$\begin{array}{r} 1. 001001010010010000000000 \text{ (1 is added before . since this is a positive number)} \\ + \quad 0.000000000001100000000000 \text{ (0 is added before . since it is a negative number)} \\ \hline \end{array}$$

$$1. 001001010011110000000000$$

The sum is normalized. There is no underflow. The final sum is

0	10001010	00100101001111000000000
----------	-----------------	--------------------------------

The result is +ve hence 0 is filled in the sign field. The exponent value of 2345.125 is copied in the exponent field of the result, since the 0.75 is adjusted to the exponent of 2345.125.

Example 2.18: Subtract $-1.00000000000000010011010x2^{-1}$ from

$1.00000000101100010001101x2^{-6}$.

$+1.00000000101100010001101x2^{-6}$

$-1.000000000000000010011010x2^{-1}$

Change the $+1.00000000101100010001101x2^{-6}$ into power of 2^{-6} .

$0.00001000000001011000100 01101x2^{-1}$

To perform subtraction take 2's complement of $-1.000000000000000010011010x2^{-1}$ which is $1 0.11111111111111101100110 x 2^{-1}$ (Here first 1 is the overflow bit).

Now add both numbers

0	0.00001000000001011000100	$01101x2^{-1}$	
1	0.11111111111111101100110		$x 2^{-1}$
	$1.00001000000001000101010$	$01101x2^{-1}$	

Floating point multiplication

The following are the steps in floating point multiplication:

1. Add the exponents
2. Multiply the significant digits
3. Normalize the product
4. Round-off the product (if necessary)

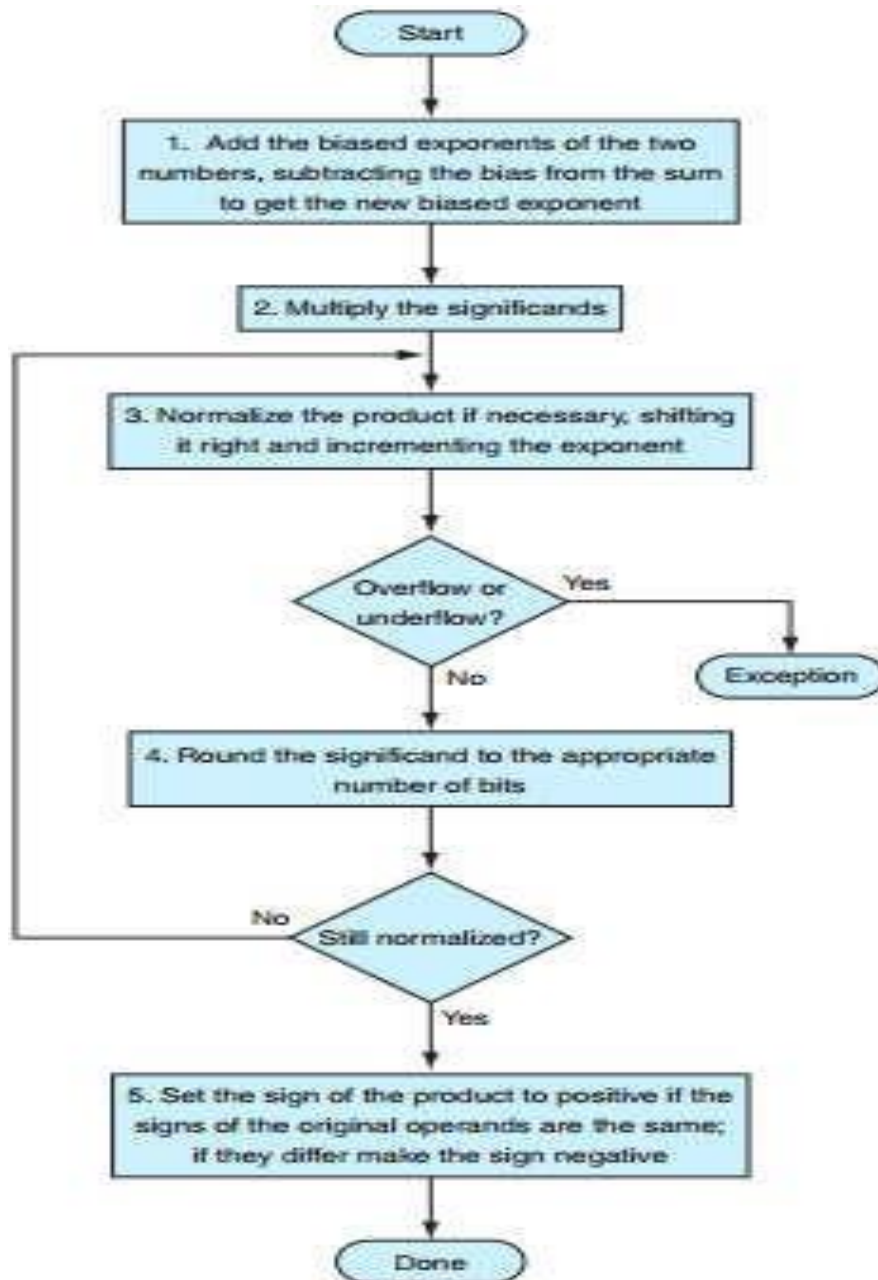


Fig 2.13: Flowchart for Floating point multiplication

Example 2.19: Multiply 1.110×10^{10} by 9.200×10^{-5} . Express the product in 3 decimal places.

1. Add the exponents

$$\text{Exponent of the product} = 10 - 5 = 5$$

2. Multiply the significant digits

$$1.110 \times 9.200 = 10.212000$$

3. Normalize the product

$$10.212 \times 10^5 = 1.0212 \times 10^6$$

4. Round-off

$$1.0212 \times 10^6 = 1.021 \times 10^6$$

Example 2.20: Perform binary multiplication on 0.5 and -0.4375.

$$0.5 = 1.000 \times 2^{-1}$$

$$0.4375 = -1.110 \times 2^{-2}$$

Add the exponents

$$\text{Exponent of the product} = -1 + -2 = -3$$

2. Multiply the significant digits

$$1.000 \times -1.110 = -1.110$$

3. Normalize the product

-1.110×10^{-3} is already normalized.

Example 2.21: Multiply $-1.110\ 1000\ 0100\ 0000\ 10101\ 0001 \times 2^{-4}$ and $1.100\ 0000\ 0001\ 0000\ 0000\ 0000 \times 2^{-2}$.

1. Add the exponents

$$\text{Exponent of the product} = -4 + -2 = -6$$

2. Multiply the significant digits

$$-1.110\ 1000\ 0100\ 0000\ 10101\ 0001 \times 1.100\ 0000\ 0001\ 0000\ 0000\ 0000$$

$$= 10.101110001111101111100110010100001000000000000$$

3. Normalize the product $1.0101110001111101111100110010100001000000000000 \times 2^{-5}$

4. Round-off (Only 23 fraction bits)

$$1.0101110001111101111100 \times 2^{-5}$$

MIPS floating point instructions

MIPS provide several instructions for floating point numbers for performing the following operations:

- Arithmetic
- Data movement (memory and registers)
- Conditional jumps

Floating Point (FP) instructions work with a different bank of registers. Registers are named \$f0 to \$f31. MIPS floating-point registers are used in pairs for double precision numbers and referred using even numbers. Single precision numbers end with .s and double precision numbers end with .d.

Category	Example	Description
FP add single	add.s \$f2, \$f4, \$f6	$f2=f4 + f6$
FP subtract single	sub.s \$f2, \$f4, \$f6	$f2=f4 - f6$
FP multiply single	mul.s \$f2, \$f4, \$f6	$f2=f4 * f6$
FP divide single	div.s \$f2, \$f4, \$f6	$f2=f4 / f6$
FP add double	add.d \$f2, \$f4, \$f6	$f2=f4 + f6$
FP subtract double	sub.d \$f2, \$f4, \$f6	$f2=f4 - f6$
FP multiply double	mul.d \$f2, \$f4, \$f6	$f2=f4 * f6$
FP divide double	div.d \$f2, \$f4, \$f6	$f2=f4 / f6$
Load word copr,1	Lwcl \$f1, 100 (\$s2)	F1=memory[s2+100]32 bit data to FP register
Store word copr,1	Swcl \$f1, 100 (\$s2)	Memory[s2+100]=f132 bit data to memory
Branch on FP true	Bclt 25	If(cond==1) goto PC+4+100PC relative branch if cond is true
Branch on FP false	Bclt 25	If(cond==0) goto PC+4+100PC relative branch if cond is false

FP compare single (eq, ne, li, le, gt, ge)	C.lt.s \$f2, \$f4	If(f2 < f4) Cond=1; else cond=0
FP compare double (eq, ne, li, le, gt, ge)	C.lt.d \$f2, \$f4	If(f2 < f4) Cond=1; else cond=0

HIGH PERFORMANCE ARITHMETIC

The performance improvement in arithmetic operations like addition, multiplication and division will increase the overall computational speed of the machine.

High performance adders

The high performance adders takes an extra input namely the transit time.

The transit time of a logical unit is used as a time base in comparing the operating speeds of different methods, and the number of individual logical units required is used in the comparison of costs.

The two multi-bit numbers being added together will be designated as A and B, with individual bits being A1, A2, B1, etc. The third input will be C. Outputs will be S (sum) R (carry), and T (transmit). The two multibit numbers being added together will be designated as A and B, with individual bits being A1, A2, B1, etc. The third input will be C. Outputs will be S (sum) R (carry), and T (transmit).

The time required to perform an addition in conventional adder is dependent on the time required for a carry originating in the first stage to ripple through all intervening stages to the S or R output of the final stage. Using the transit time of a logical block as a unit of time, this amounts to two levels to generate the carry in the first stage, plus two levels per stage for transit through each intervening stage, plus two levels to form the sum in the final stage, which gives a total of two times the number of stages.

$$C_n = R_{n-1}$$

$$C_n = D_{n-1} \parallel T_{n-1} R_{n-2}$$

$$C_n = D_{n-1} \parallel T_{n-1} D_{n-2} \parallel T_{n-1} T_{n-2} R_{n-3}$$

By allowing n to have successive values starting with one and omitting all terms containing a resulting negative subscript, it may be seen that each stage of the adder will

require one OR stage with n inputs and n AND circuits having one through n inputs, where n is the position number of the particular stage under consideration.

High performance Multiplication

Multiplication using variable length shift

- The multiplier and the partial product will always be shifted the same amount and at the same time.
- The multiplier is shifted in relation to the decoder, and the partial product with relation to the multiplicand.
- Operation is assumed starting at the low-order end of the multiplier, which means that shifting is to the right.
- If the lowest-order bit of the multiplier is a one, it is treated as though it had been approached by shifting across zeros.

Rules:

1. When shifting across zeros (from low order end of multiplier), stop at the first one.
 - a) If this one is followed immediately by a zero, add the multiplicand, then shift across all following zeros.
 - b) If this one is followed immediately by a second one, subtract the multiplicand, then shift across all following ones.
 2. When shifting across ones (from low order end of multiplier), stop at the first zero.
 - a) If this zero is followed immediately by a one, subtract the multiplicand, then shift across all following ones.
 - b) If this zero is followed immediately by a second zero, add the multiplicand, then shift across all following zeros.
- A shift counter or some equivalent device must be provided to keep track of the number of shifts and to recognize the completion of the multiplication.

- If the high-order bit of the multiplier is a one and is approached by shifting across ones, that shift will be to the first zero beyond the end of the multiplier, and that zero along with the bit in the next higher order position of the register will be decoded to determine whether to add or subtract.
- For this reason, if the multiplier is initially located in the part of the register in which the product is to be developed, it should be so placed that there will be at least two blank positions between the locations of the low-order bit of the partial product and the high-order bit of the multiplier.
- Otherwise the low-order bit of the product will be decoded as part of the multiplier.

Multiplication Using Uniform Shifts

- Multiplication which uses shifts of uniform size and permits predicting the number of cycles that will be required from the size of the multiplier is preferable to a method that requires varying sizes of shifts.
- The most important use of this method is in the application of carry-save adders to multiplication although it can also be used for other applications.

Uniform shifts of two

- Assume that the multiplier is divided into two-bit groups, an extra zero being added to the high-order end, if necessary, to produce an even number of bits.
- Only one addition or subtraction will be made for each group, and, using the position of the low-order bit in the group as a reference, this addition or subtraction will consist of either two times or four times the multiplicand.
- These multiples may be obtained by shifting the position of entry of the multiplicand into the adder one or two positions left from the reference position.
- The last cycle of the multiplication may require special handling.
- Following any addition or subtraction, the resulting partial product will be either correct or larger than it should be by an amount equal to one times the multiplicand.
- Thus, if the high-order pair of bits of the multiplier is 00 or 10, the multiplicand would be multiplied by zero or two and added, which gives a correct partial product.
- If the high-order pair of bits is 01 or 11, the multiplicand is multiplied by two or four,

not one or three, and added. This gives a partial product that is larger than it should be, and the next add cycle must correct for this.

- Following the addition the partial product is shifted left- two positions. This multiplies it by four, which means that it is now larger than it should be by four times the multiplicand.
- This may be corrected during the next addition by subtracting the difference between four and the desired multiplicand multiple.
- Thus, if a pair ends in zero, the resulting partial product will be correct and the following operation will be an addition.
- If a pair ends in a one, the resulting partial product will be too large, and the following operation will be a subtraction.
- It can now be seen that the operation to be performed for any pair of bits of the multiplier may be determined by examining that pair of bits plus the low-order bit of the next higher-order pair.
- If the bit of the higher-order pair is a zero, an addition will result; if it is one, a subtraction will result. If the low-order bit of a pair is considered to have a value of one and the high-order bit a value of two, then the multiple called for by a pair is the numerical value of the pair if that value is even and one greater if it is odd.
- If the operation is an addition, this multiple of the multiplicand is used. If the operation is a subtraction (the low-order bit of the next higher order pair a one), this value is combined with minus four to determine the correct multiple to use.
- The result will be zero or negative, with a negative result meaning subtract instead of add.

Multiplication Using Carry-Save Adders

- When successive additions are required before the final answer is obtained, it is possible to delay the carry propagation beyond one stage until the completion of all of the additions, and then let one carry-propagate cycle suffice for all the additions. Adders used in this manner are called **carry-save adders**.
- A carry-save adder consists of a number of stages, each similar to the full adder. It differs from the ripple-carry adder in that the carry (R) output is not connected directly

to the next-higher-order stage of the same adder, but goes to an intermediate register or other device in the same manner as the sum (S) output.

- A carry-save adder has three inputs which, as far as use is concerned, may be considered identical, and two outputs which are not identical and must be treated in different manners.
- The procedure for adding several binary numbers by using a carry-save adder would be as follows.
- Designate the inputs for the n th bit as A_n , B_n , and C , and the outputs for the same bit as S_n and R , where S_n is the sum output and R is the carry output.
- In the first cycle enter three of the input numbers into A , B , and C .
- In the second cycle enter the S and R obtained from the previous cycle into A and B and the fourth input number into C .
- In this operation S_n goes into A_{n+1} , but R_n goes into B_{n+1} , where B_{n+1} is in the next higher-order bit position than B .
- This is continued until all of the input numbers have been entered into the adder.
- Each add cycle advances all carries one position, add cycles as already described may be continued with zeros being entered into the third input each time until the R outputs of all stages become zero.
- The alternative is to enter S and R into a carry-propagate adder and allow time for one cycle through it.
- This carry-propagate adder may be completely separate from the carry-save unit, or it may be a combined unit with a control line for selecting either carry-save or carry-propagate operation.

SUB WORD PARALLELISM

*A subword is a lower precision unit of data contained within a word.
In subword parallelism, multiple subwords are packed into a word and
then process whole words.*

With the appropriate subword boundaries this technique results in parallel processing of subwords. Since the same instruction is applied to all subwords within the word, This is a

form of SIMD(Single Instruction Multiple Data) processing. It is possible to apply subword parallelism to noncontiguous subwords of different sizes within a word. In practical implementation is simple if subwords are same size and they are contiguous within a word. The data parallel programs that benefit from subword parallelism tend to process data that are of the same size.

Example: If word size is 64bits and subwords sizes are 8,16 and 32 bits. Hence an instruction operates on eight 8bit subwords, four 16bit subwords, two 32bit subwords or one 64bit subword in parallel.

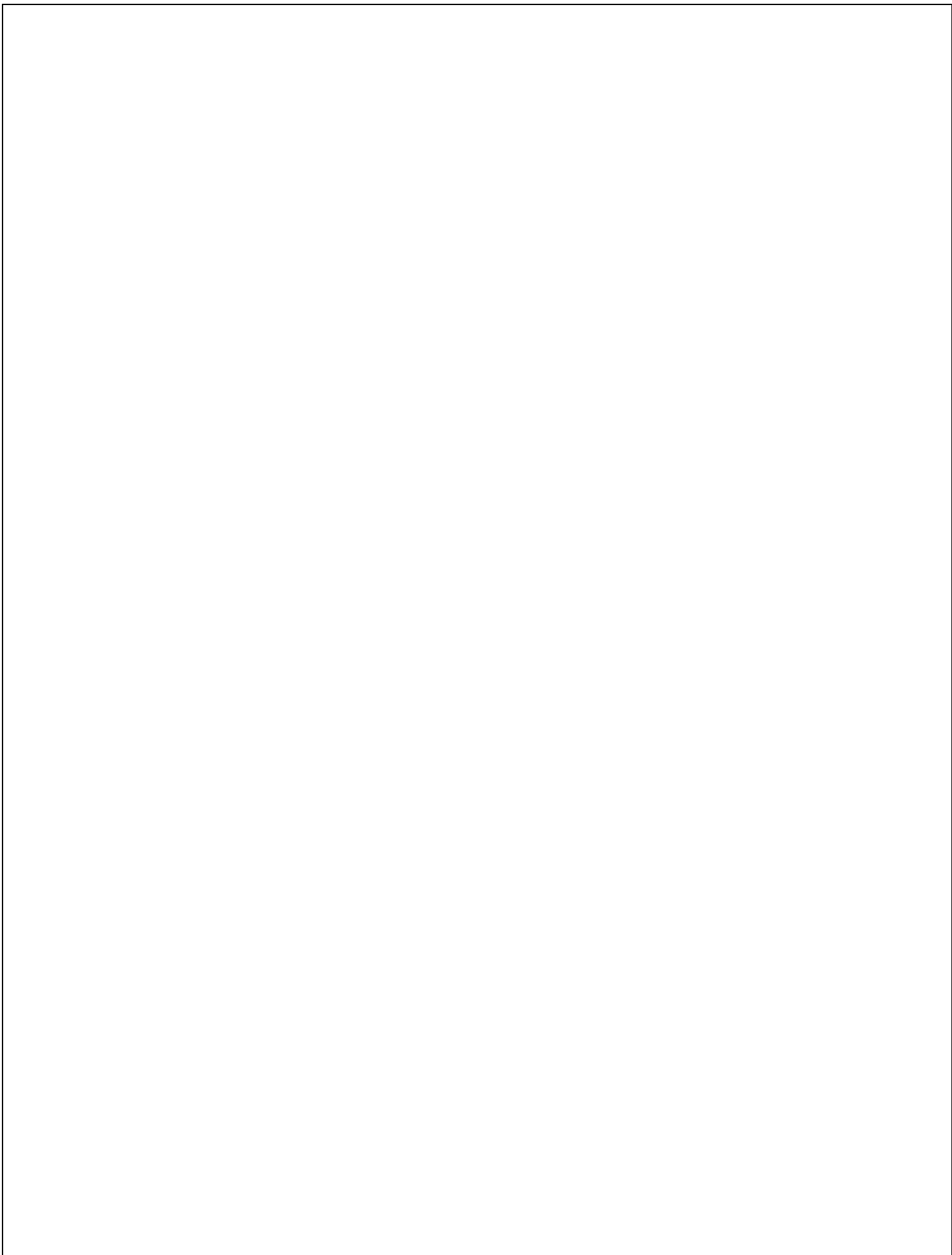
Advantages of subword parallelism

- Subword parallelism is an efficient and flexible solution for media processing because algorithms exhibit a great deal of data parallelism on lower precision data.
- It is also useful for computations unrelated to multimedia that exhibit data parallelism on lower precision data.
- Graphics and audio applications can take advantage of performing simultaneous operations on short vectors.
- One key advantage of subword parallelism is that it allows general-purpose processors to exploit wider word sizes even when not processing high-precision data.
- The processor can achieve more subword parallelism on lower precision data rather than wasting much of the word-oriented data paths and registers.

Support for subword parallelism

- Data-parallel algorithms with lower precision data map well into subword-parallel programs.
- The support required for such subword-parallel computations then mirrors the needs of the data-parallel algorithms.
- To exploit data parallelism, we need subword parallel compute primitives, which perform the same operation simultaneously on subwords packed into a word.
- These may include basic arithmetic operations like add, subtract, multiply, divide, logical, and other compute operations.

- Data-parallel computations also need
 1. Data alignment before or after certain operations for subwords representing fixed-point numbers or fractions
 2. Subword rearrangement within a register so that algorithms can continue parallel processing at full clip
 3. A way to expand data into larger containers for more precision in intermediate computations. Similarly, a way to contract it to a fewer number of bits after the computation's completion and before its output.
 4. Conditional execution
 5. Reduction operations that combine the packed subwords in a register into a single value or a smaller set of values.
 6. A way to clip higher precision numbers to fewer bits for storage or transmission.
 7. The ability to move data between processor registers and memory, as well as the ability to loop and branch to an arbitrary program location.



UNIT - III

THE PROCESSOR

INTRODUCTION

The key performance metrics of the computer systems are;

- i. Instruction count: This depends on the compiler used and instruction set architecture.
- ii. Clock cycle time: This depends on processor implementation.
- iii. Clock cycles per instruction (CPI): This depends on processor implementation.

MIPS ARCHITECTURE

MIPS (Million Instructions Per Second) is a simple, streamlined, highly scalable RISC architecture with adopted by the industries.

The features that makes its widely useable are:

- Simple load and store with large number of register
- The number and the character of the instructions
- Better pipelining efficiency with visible pipeline delay slots
- Efficiency with compilers

These features make the MIPS architecture to deliver the highest performance with high levels of power efficiency. It is important to learn the architecture of MIPS to understand the detailed working of the processors.

Implementation of MIPS

MIPS has 32 General purpose registers (GPR) or integer registers (64 bit) holding integer data. Floating point registers (FPR) are also available in MIPS capable of holding both single precision (32 bit) and double precision data (64 bit). The following are the data types available for MIPS:

Size	Name	Registers
8 bits	Byte	Integer register
16 bits	Half word	Integer register
32 bits	Word	Floating point register
64 bits	Double word	Floating point register

With these resources the MIPS performs the following operations:

- Memory referencing: load word (lw) and store word (sw)
- Arithmetic-logical instructions: add, sub, and, or, and slt
- Branch instructions: equal (beq) and jump (j)

The common steps in load and store instructions are:

- i. Set the program counter (PC) to the address of the code and fetch the instruction from that memory.
- ii. Read one or two registers, using fields of the instruction to select the registers to read. For the load word instruction, read only one register and for store word the processor has to operate on two registers.

The ALU operations are done and the result of the operation is stored in the destination register using store operation. When a branching operation is involved, then next address to be fetched must be changes based on the branch target.

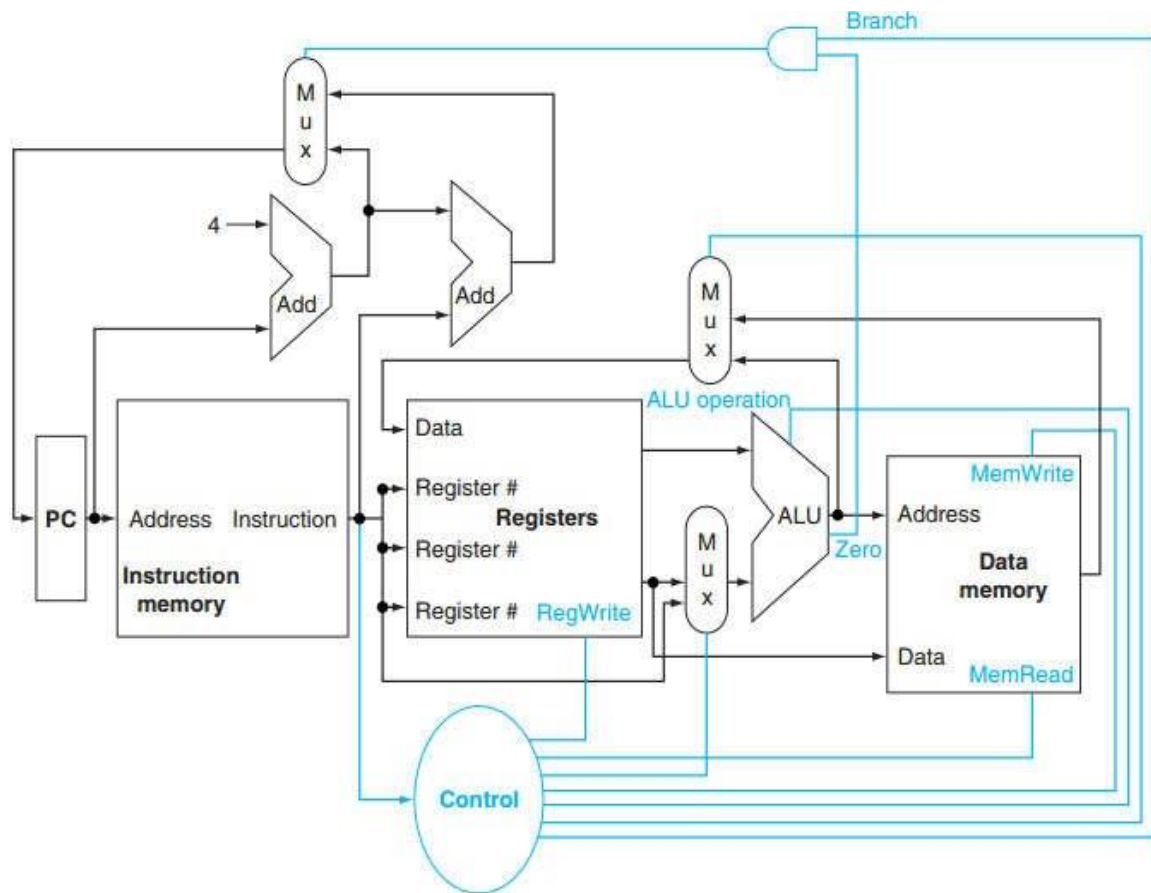


Fig 3.1 :Implementation of MIPS architecture with multiplexers and control lines

Sequence of operations

- **Program Counter (PC):** This register contains the address (location) of the instruction currently getting executed. The PC is incremented to read the next instruction to be executed.
- The operands in the instruction are fetched from the registers.
- The ALU or branching operations are done. The results of the ALU operations are stored in registers. If the result is given in load and store forms, then the results are written to the memory address and from there they are transferred to the registers.
- In case of branch instructions, the result of the branch operation is used to determine the next instruction to be executed.

- The multiplexer (MUX1), selects one input control line from multiple inputs. This acts as a **data selector**.
- This helps to control several units depending on the type of instruction.
- The top multiplexor controls the target value of the PC. To execute next instruction the PC is set as PC+4. To execute a branch instruction set the PC to the branch target address.
- The multiplexor is controlled by the AND gate that operates on the zero output of the ALU and a control signal that indicates that the instruction is a branch.
- The multiplexor (MUX2) returns the output to the register file for loading the resultant data of ALU operation into the registers.
- MUX3 determines whether the second ALU input is from the registers or from the offset field of the instruction.
- The control lines determine the operation performed at the ALU. The control lines decide whether to read or write the data.

MIPS instruction format

There are only three instruction formats in MIPS. The instructions belong to any one of the following type:

- Arithmetic/logical/shift/comparison
- Control instructions (branch and jump)
- Load/store
- Other (exception, register movement to/from GP registers, etc.)

All the instructions are encoded in one of the following three formats:

I type: Load and store instructions

Opcode	Rs	Rt	Immediate
--------	----	----	-----------

R-type: Register to register operations

Opcode	Rs	Rt	Rd	Shamt	Funct
--------	----	----	----	-------	-------

J-Type: Jump instructions

Opcode	Offset
--------	--------

The data and memory are well separated in MIPS implementation because:

- The instruction formats for the operations are not unique; hence the memory access will also be different.
- Maintaining separate memory area is less expensive.
- The operations of the processor are performed in single cycle. A single memory (for both data and memory access) will not allow for two different accesses within one cycle.

LOGIC DESIGN CONVENTIONS

The information in a computer system is encoded in binary form (0 or 1). The high voltage is encoded as 1 and low voltage as 0. The data is transmitted inside the processors through control wires / lines. These lines are capable of carrying only one bit at a time. So transfer of multiple data can be done through deploying multiple control lines or buses. The data should be synchronised with time by transferring it according to the clock pulses. All the internal operations inside the processor are implemented through logic elements. The logic elements are broadly classified into: **Combinatorial and Sequential elements**.

Differences between Combinatorial and Sequential elements

Combinatorial Elements	Sequential Elements
The output of the combinatorial circuit depends only on the current input.	The output depends on the previous stage outputs.
It has faster operation speed and easy implementation.	It has comparatively low operation speed and tough implementation.
No feedback connections.	The output is connected with the input through feedback connections.
For a given set of inputs, combinatorial elements give the same output since there is no storage of past data.	The outputs vary based on previous outputs.

The basic building blocks are gates, which are time independent.	The basic building blocks are flip flops, which are time dependent.
It is used for Arithmetic and Logic operations.	It is used for data storage.
No need for trigger.	Triggering is needed to control the clock cycles.
No memory element.	Memory element is needed which is used to store the states.
Eg: Encoder, full adder, Decoder, Multiplier	Eg: Counters

Importance of state elements

The state elements characterise the machines. They contain state or status values so that the machine can be restored with the previous values by retaining the values in the state element. A state element has at least two inputs and one output. The required inputs are the data value to be written into the element and the clock, which determines when the data value is written. The output from a state element provides the value that was written in an earlier clock cycle. The following are the state elements in Fig 3.1: instructions, memories and registers.

Clocking Methodology

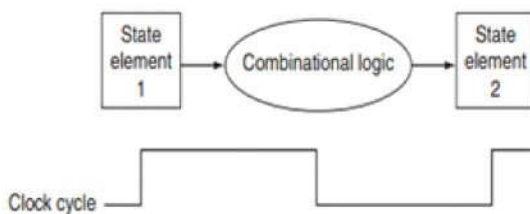


Fig 3.2 a: Combinatorial Logic

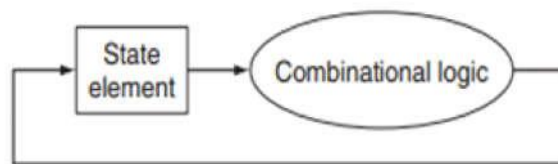


Fig 3.2 b: Edge triggered Logic

A clocking methodology is a set of rules for interconnecting components and clock signals that, when followed, guarantee proper operation of the resulting system.

The primary objective of clocking methodology is timing correlation.

Edge triggered clocking methodology

This allows the processor to read the register contents, send the value through some combinatorial logic and write that register in same clock cycle under the assumption that the state elements are controlled by implicit clock cycles.

- Here, the stored values are updated only on a clock edge.
- In combinatorial logic, the input must be read, processed and the output must be sent to the location, all in one single clock cycle (Fig 3.2 a).
- The driving force of this combinatorial circuit will be an explicit control signal.
- All the changes occur only when the clock signal is triggered.
- In edge-triggered methodology, the contents of a register are read and the value is sent through combinatorial logic, and written to that register in the same clock cycle.
- This prevents the access of inconsistent intermediate data
- Feedback cannot occur within 1 clock cycle because of the edge-triggered update of the state element.
- The clock cycle still must be long enough so that the input values are stable when the active clock edge occurs.

BUILDING A DATAPATH

A datapath is a representation of the flow of information (data and instructions) through the CPU, implemented using combinatorial and sequential circuitry.

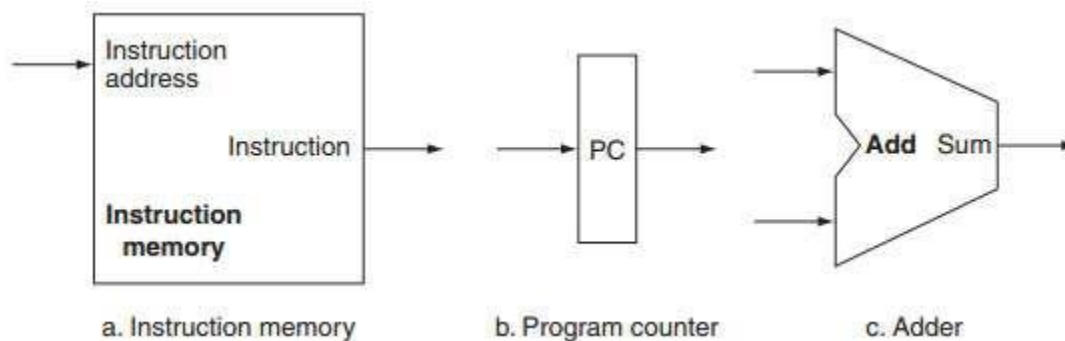


Fig: 3.3 Components of Datapath

Datapath is a functional unit that operates or hold data. In the MIPS implementation the datapath elements includes instruction and data memories, the register file, the arithmetic logic unit (ALU), and adders. The functionalities of basic elements are listed below:

- **Instruction Memory:** It is a state element that provides read access because the datapath do not perform write operation. This combinatorial memory always holds contents of location specified by the address.
- **Program Counter (PC):** This is a 32 bit state register containing the address of the current instruction that is being executed. It is updated after every clock cycle and do not require an explicit write signal.
- **Adder:** This is a combinatorial circuit that updates the value of PC after every clock cycle to get that address of the next instruction to be executed.

Instruction Fetch:

The fundamental operation in Instruction Fetch is to send the address in the PC to the instruction memory and obtain the specified instruction, and the increment the PC.

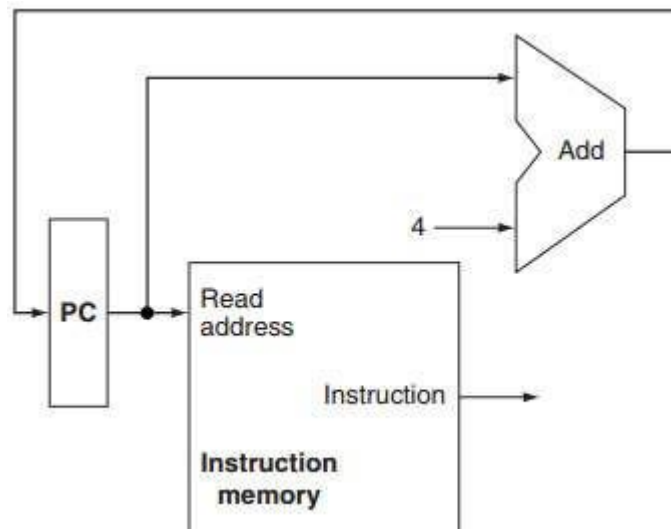


Fig: 3.4: Instruction Fetch

R type instructions:

- They all read two registers, perform an ALU operation on the contents of the registers, and write the result.
- This instruction class includes add, sub, and, or, and slt.
- The processor's 32 general-purpose registers are stored in a structure called **register file**.
- A register file is a collection of registers in which any register can be read or written by specifying the number of the register in the file. The register file contains the register state of the machine.
- The R-format always performs ALU operation that has three register operands (2-read and 1-write).
- The register number must be specified in order to read the data from the register file. Also the output from a register file will contain the data that is read from the register.
- The write operation to a register has two inputs: the register number and the value to be written. This operation is edge triggered.

Load and Store instructions:

- The load and store instructions compute a memory address by adding the base register.
- If the instruction is a load, the value read from memory must be written into the register file in the specified register.
- The memory is computed by adding the address of base register and the 16-bit signed offset field (which is a part of the instruction).
- If the instruction is a store, the value to be stored must also be read from the register.

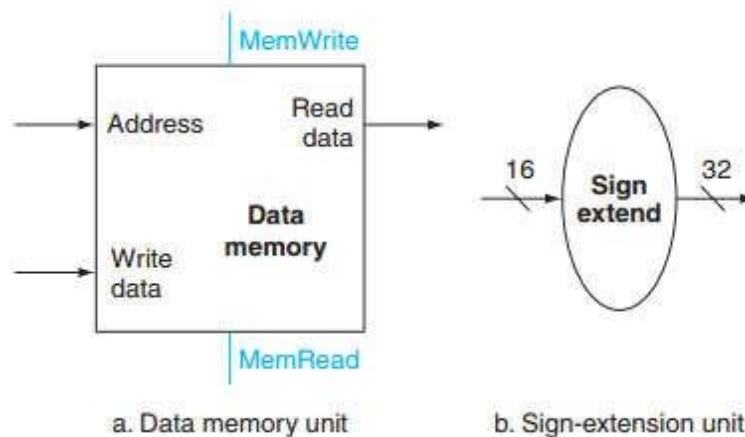


Fig 3.5: Data memory and sign extension unit

- The processor has a sign extension unit to **sign-extend** the 16-bit offset field in the instruction to a 32-bit signed value.
- The data memory unit is necessary to perform write operation of store instruction. So it has both read and write control signals, an address input and data input.

Branch Instructions:

Branch Target is the address specified in a branch, which is used to update the PC if the branch is taken. In the MIPS architecture the branch target is computed as the sum of the offset field of the instruction and the address of the instruction following the branch.

- The beq instruction (branch instruction) has three operands, two registers that are compared for equality, and a 16-bit offset to compute the branch target address.
beq t1, t2, offset
- Thus, the branch datapath must do two operations: compute the branch target address and compare the register contents.
- **Branch Taken** is where the branch condition is satisfied and the program counter (PC) loads the branch target. All unconditional branches are taken branches.
- **Branch not Taken** is where the branch condition is false and the program counter (PC) loads the address of the instruction that sequentially follows the branch.
- The branch target is calculated by taking the address of the next instruction after the branch instruction, since the PC value will be updated as PC+4 even before the branch decision is taken.
- The offset field is shifted left 2 bits to increase the effective range of the offset field by a factor of four.

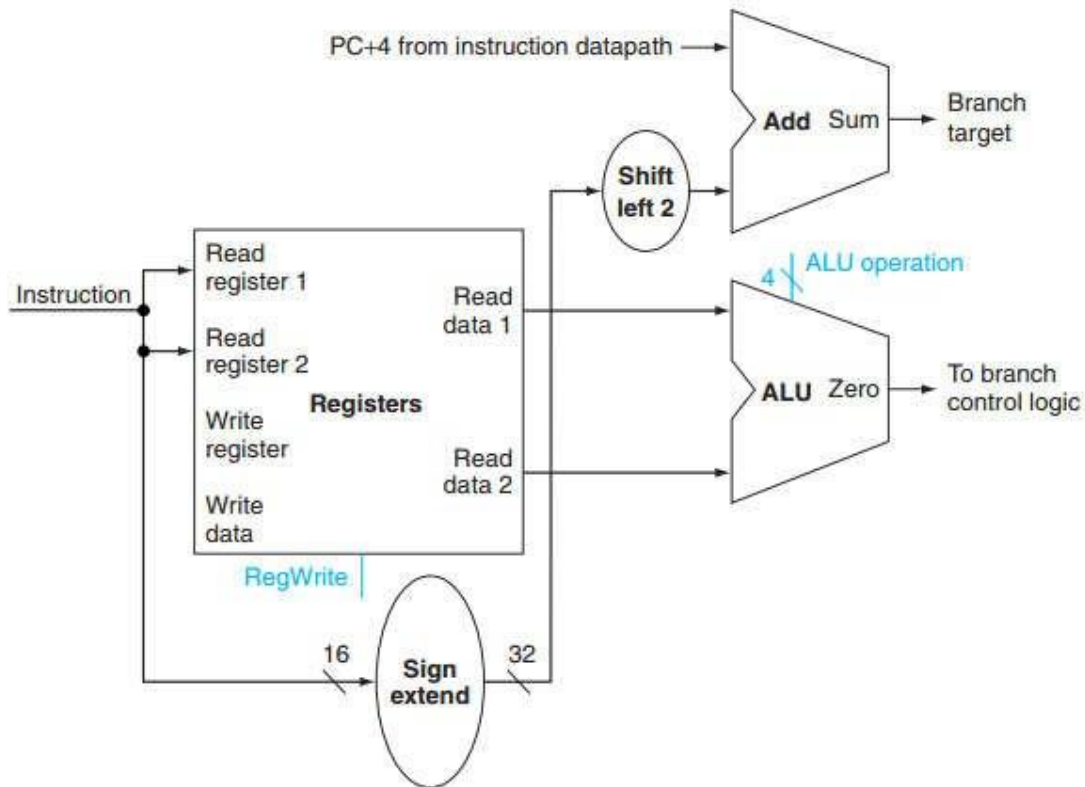


Fig 3. 6: Datapath of branch Instructions

- The unit labelled Shift left 2 adds two zero's to the low-order end of the sign-extended offset field. This operation truncated the sign values.
- The control logic decides whether the incremented PC or branch target should replace the PC, based on the Zero output of the ALU.
- The jump instruction operates by replacing the lower 28 bits of the PC with the lower 26 bits of the instruction shifted left by 2 bits. This shift is done by concatenating 00 to the jump offset.
- **Delayed branch** is where the instruction immediately following the branch is always executed, independent of whether the branch condition is true or false.
- MIPS architecture implements delayed branch (i.e.) the instruction immediately following the branch is always executed, independent of whether the branch condition is true or false.

- When the condition is false, the execution looks like a normal branch.
- When the condition is true, a delayed branch first executes the instruction immediately following the branch in sequential instruction order before jumping to the specified branch target address.
- Delayed branches facilitate pipelining.

Creating a single Datapath

- A simple implementation of a single datapath is to execute all operations within one clock cycle.
- The datapath resources can be utilized only for one clock cycle. To facilitate this, some resources must be duplicated for simultaneous access while other resources will be shared .
- One example is having separate memory for instructions and memory.
- When a resource is used in shared mode, then multiple connections must be made. The selection of which control will access the resource will be decided by a multiplexer.

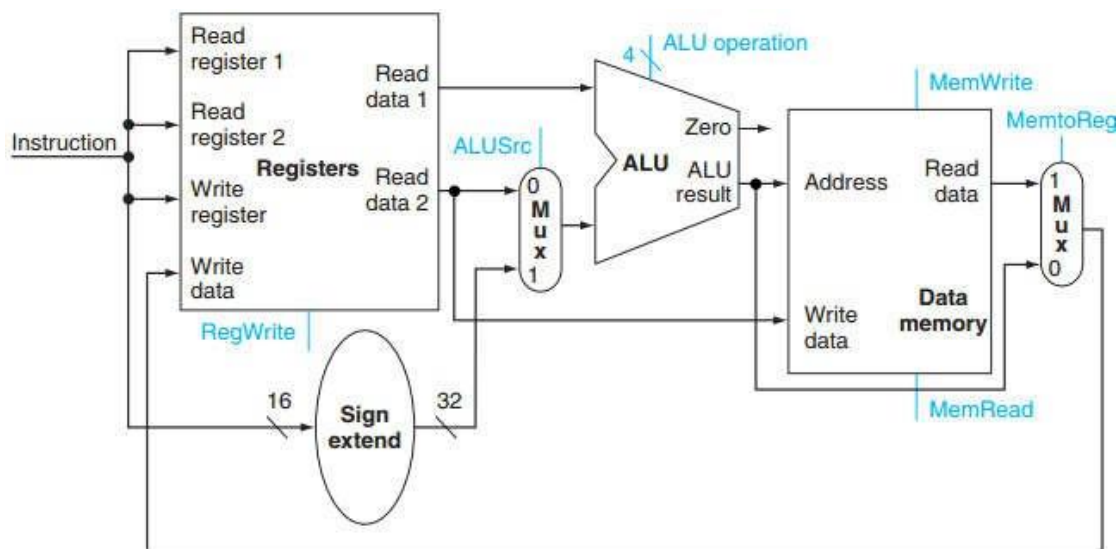


Fig. 3.7: Simple datapath

- The datapath illustrated in Fig 3.7 shows the assembling of individual elements into a simple datapath.

- To implement branch instructions the datapath must include an adder circuitry to compute branch target (Refer Fig: 3.6).
- The control unit for this datapath must take inputs and generate a write signal for each state element. Apart from the inputs a selector control must be included for each multiplexor and the ALU control.
- The operations of arithmetic-logical (or R-type) instructions and the memory instructions datapath are almost similar.
- The arithmetic-logical instructions use the ALU with the inputs coming from the two registers. The memory instructions can also use the ALU to do the address calculation, but the second input is the sign-extended 16-bit offset field from the instruction.

SIMPLE IMPLEMENTATION SCHEME

The basic implementation includes a subset of the core MIPS instruction set:

- The memory-reference instructions load word (lw) and store word (sw).
- The arithmetic-logical instructions add, sub, AND, OR, and slt.
- The instructions branch equal (beq) and jump (j).

For any instruction, the following two steps are same:

1. Send the program counter (PC) to the memory that contains the code and fetch the instruction from that memory.
2. Read one or two registers, using fields of the instruction to select the registers to read.

Load instruction needs to read only one register, but most other instructions require reading two registers. The remaining actions required to complete the instruction depend on the instruction class. For the three instruction classes namely memory-reference, arithmetic-logical, and branches, the actions are mostly the same. This is due to the simplicity and regularity of the MIPS instruction set.

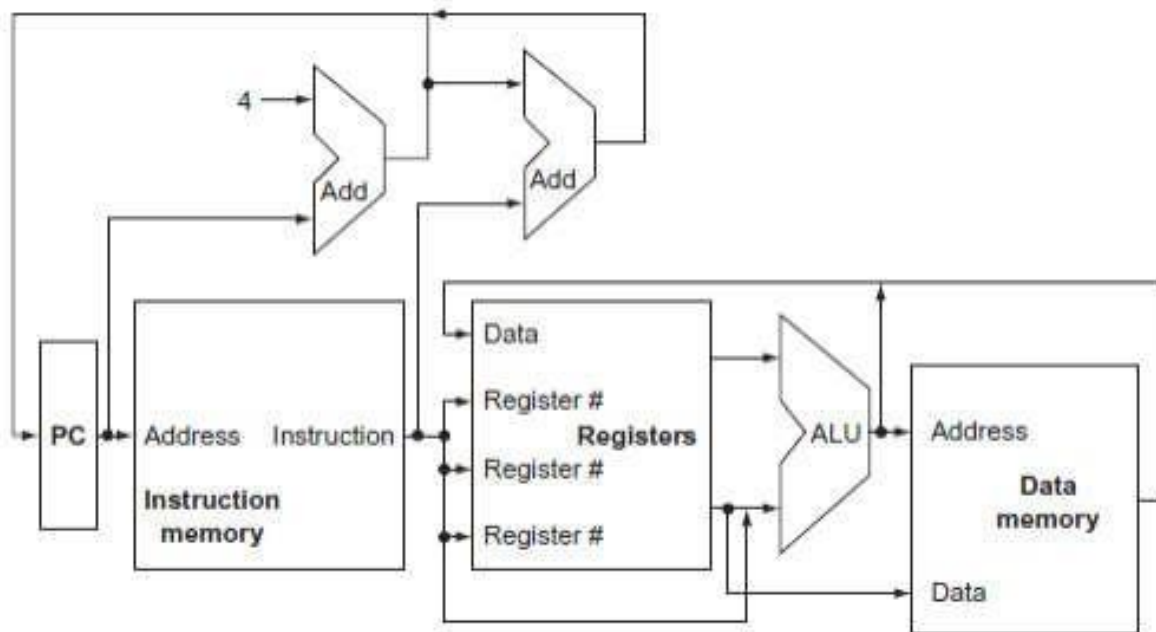


Fig 3.8: An abstract view of MIPS implementation

Instruction Formats of MIPS

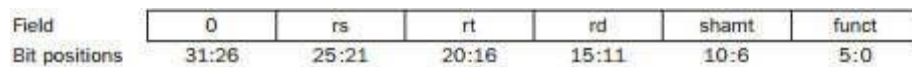


Fig 3.9: R-Format Instruction

Instruction format for R-format instructions have an opcode of 0. These instructions have three register operands: sources: rs, rt, and destination: rd. The ALU function is in the funct field and is decoded by the ALU control design in the previous section. This instruction type is used to implement are add, sub, and, or, and slt. The shamt field is for shifting operation.

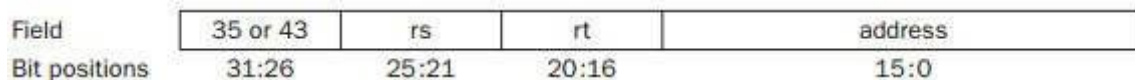


Fig 3.10: Load or store instruction

Instruction format for load specified by opcode = 35ten and store is specified by opcode = 43ten) instructions. The register *rs* is the base register that is added to the 16-bit address field to form the memory address. For loads, *rt* is the destination register for the loaded value. For stores, *rt* is the source register whose value should be stored into memory.

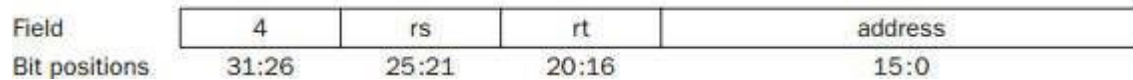


Fig 3.11: Branch Instructions

Instruction format for branch equal (opcode = 4). The registers *rs* and *rt* are the source registers that are compared for equality. The 16-bit address field is sign extended, shifted, and added to the PC to compute the branch target address.

- All instruction classes, except jump, use the arithmetic-logical unit (ALU) after reading the registers.
- The memory-reference instructions use the ALU for an address calculation, the arithmetic-logical instructions for the operation execution, and branches for comparison.
- After using the ALU, the actions required to complete various instruction classes differ.
- A memory-reference instruction will need to access the memory either to read data for a load or write data for a store.
- An arithmetic-logical or load instruction must write the data from the ALU or memory back into a register.
- Branch instruction need to change the next instruction address based on the comparison; otherwise, the PC should be incremented by 4 to get the address of the next instruction.
- All instructions start by using the program counter to supply the instruction address to the instruction memory.
- After the instruction is fetched, the register operands used by an instruction are specified by fields of that instruction.

- Once the register operands have been fetched, they can be operated on to compute a memory address (for a load or store), to compute an arithmetic result (for an integer arithmetic-logical instruction), or a compare (for a branch).
- If the instruction is an arithmetic-logical instruction, the result from the ALU must be written to a register.
- If the operation is a load or store, the ALU result is used as an address to either store a value from the registers or load a value from memory into the registers.
- The result from the ALU or memory is written back into the register file.
- Branches require the use of the ALU output to determine the next instruction address, which comes either from the ALU (where the PC and branch offset are summed) or from an adder that increments the current PC by 4.
- The thick lines interconnecting the functional units represent buses, which consist of multiple signals.
- Fig.3.12 shows the data path of Fig 3.8 with the three required multiplexors added, and control lines for the major functional units.
- A control unit, which has the instruction as an input, is used to determine how to set the control lines for the functional units and two of the multiplexors.
- The third multiplexor, which determines whether $PC + 4$ or the branch destination address is written into the PC, is set based on the Zero output of the ALU, which is used to perform the comparison of a beq instruction.

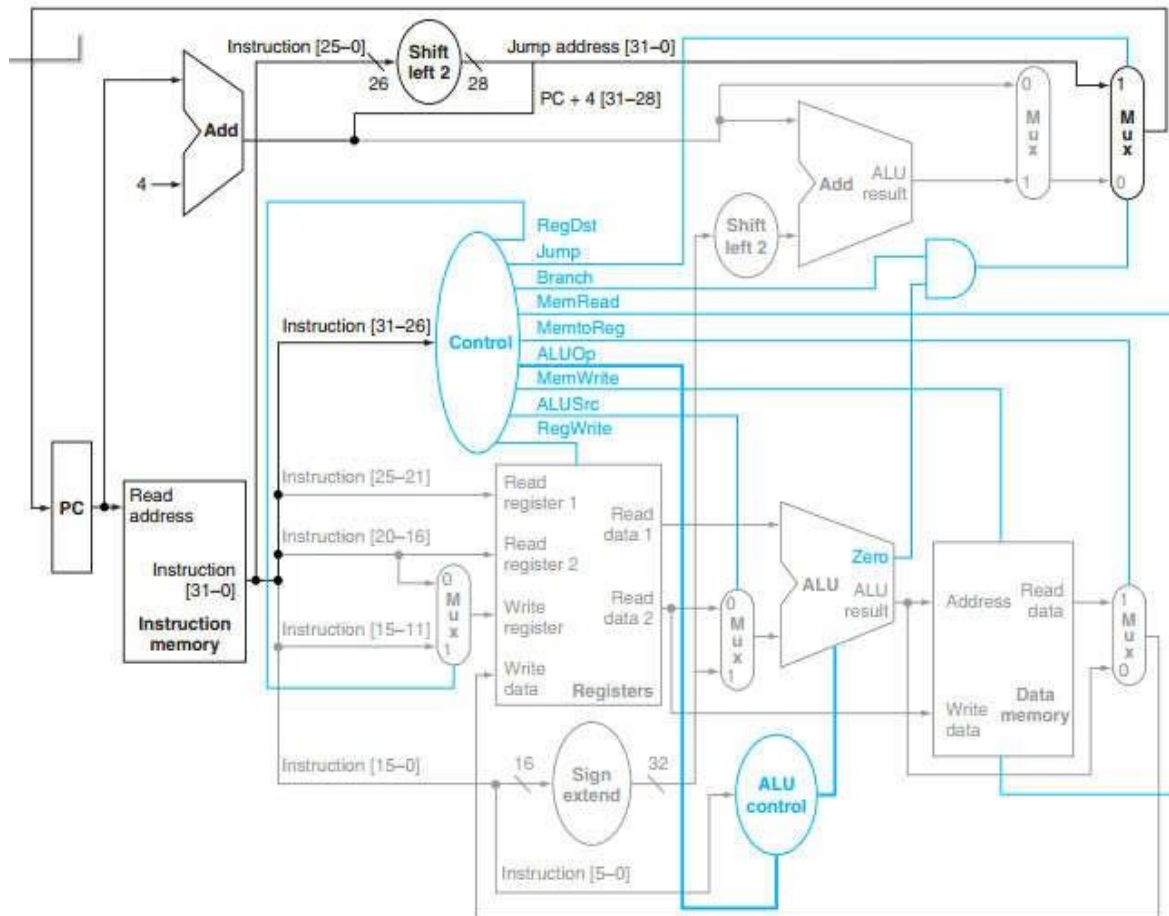


Fig 3.12: Implementation scheme with control lines

Operation of the Datapath given in Fig 3.12:

Four steps to execute the instruction; these steps are ordered by the flow of information:

1. The instruction is fetched, and the PC is incremented.
2. Two registers, \$t2 and \$t3, are read from the register file. the main control unit computes the setting of the control lines during this step.
3. The ALU operates on the data read from the register file, using the function code (bits 5:0, which is the funct field, of the instruction) to generate the ALU function.
4. The result from the ALU is written into the register file using bits 15:11 of the instruction to select the destination register (\$t1).

Effect of the control signals

Signal	When deasserted	When asserted
RegDst	The register destination number for Write register comes from the rt field (bits 20:16)	The register destination number for Write register comes from the rd field (bits 15:11)
RegWrite	None	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second operand comes from the second register file output (Read data 2).	The second operand is the sign extended lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder after computing PC+4	The PC is replaced by the output of the adder after computing the branch target.
MemRead	None	Data memory contents designated by the address input are placed on the Read data input.
MemtoReg	The value given to Write data input is got from the ALU.	The value given to Write data input is got from the data memory.

The setting of the control lines is completely determined by the opcode fields of the instruction as given below:

Instruction	Reg Dst	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALU Op1	ALU Op0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	x	1	x	0	0	1	0	0	0
beq	x	0	x	0	0	0	1	0	1

Finalizing the Controls

The logic values for a comprehensive control unit can be expressed as a single large truth table. This table combines all the outputs and uses the opcode bits as inputs. It completely specifies the control function.

Input / Output	Signal	R-format Name	Lw	Sw	Beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	x	x
	ALUSrc	0	1	1	0
	MemtoReg	0	1	x	x
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

PIPELINING

Pipelining is an implementation technique in which multiple instructions are executed simultaneously by overlapping them in execution to save time and resource. The previous instruction will be in the execution phase when the current instruction is fetched from the memory.

Need for Pipelining

Without a pipeline, a computer processor fetches the first instruction from memory, performs the operation mentioned in it, and then goes to fetch the next instruction from memory. While fetching the instruction, the arithmetic unit of the processor is idle. It must wait until it is loaded with next instruction.

With pipelining, the computer architecture allows the next instructions to be fetched while the processor is performing arithmetic operations, holding them in a buffer close to the processor. The result is an increase in the number of instructions that can be performed during a given time period.

Stages in MIPS pipelining:

The following are the various stages in pipelining:

1. **Instruction Fetch (IF):**Fetch instruction from memory.
2. **Instruction Decode (RD):**Read registers while decoding the instruction. The format of MIPS instructions allows reading and decoding to occur simultaneously.
3. **Execute:**Execute the operation or calculate an address. This involves ALU operations.
4. **Memory access (MEM):**Access an operand in data memory.
5. **Write Back (WB):**Write the result into a register.

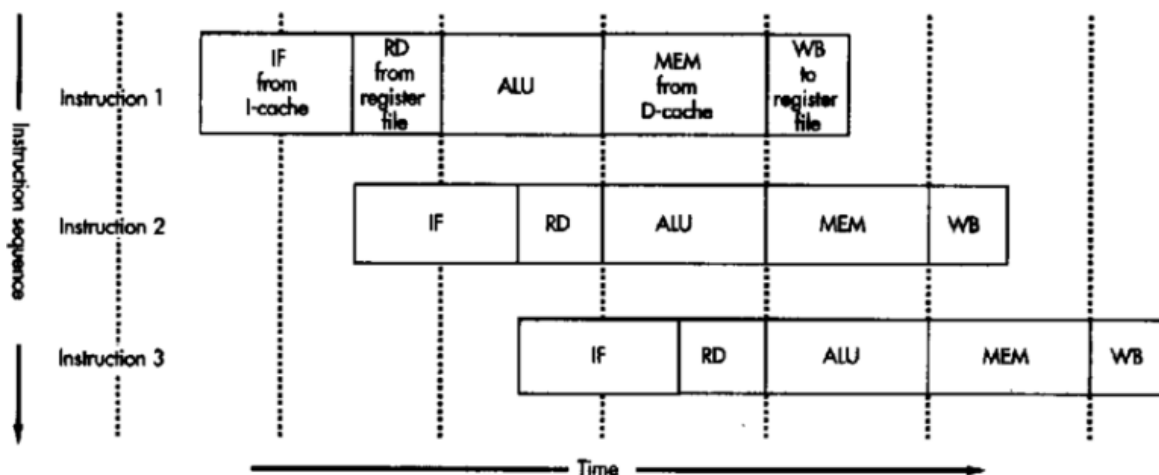


Fig 3.13: 5 stage pipelining of MIPS architecture

The pipelining speed can be manipulated using the expression:

$$\text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instruction}_{\text{nonpipelined}}}{\text{Number of pipe stages}}$$

Pipelining improves performance by increasing instruction throughput. It is not decreasing the execution time of an individual instruction, but increases the number of instructions that complete its execution for a given time period. Thus the overall performance of the processor is improved both in terms of resource utilization and throughput.

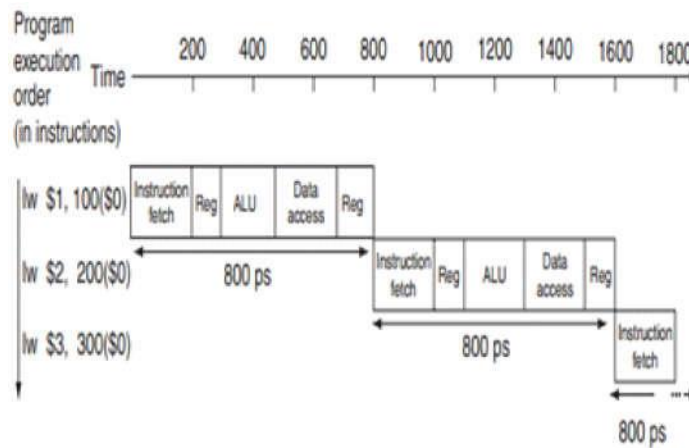


Fig 3.14 a) Non pipelined Execution

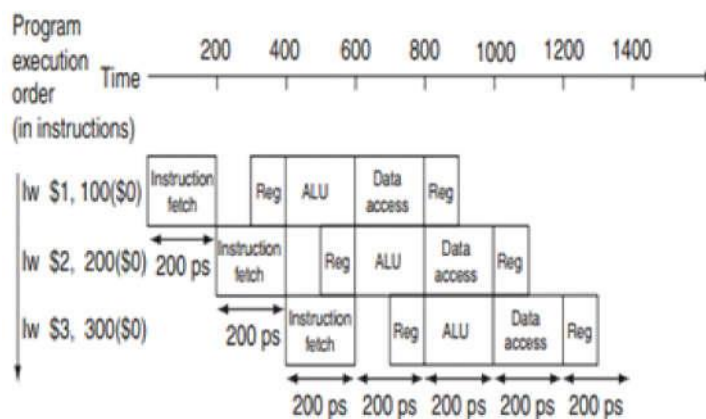


Fig 3.14 b) Pipelined Execution

Fig 3.14 shows the comparison of execution of instructions with and without pipelining on same hardware components. The timeline clearly indicates that there is a difference in execution time and resource utilization. The challenges in implementing pipelining may arise due to slowest resource.

Designing instruction sets for Pipelining

- The simplicity and generality of MIPS instructions are that they are of same length. This facilitates easy instruction fetching in the first stage of pipelining.
- MIPS has only a few instruction formats. In every instruction format, the source operand register is located at the same position in the instruction format.
- This symmetry eases the instruction decode stage by reading the register file simultaneously while the hardware is determining the type of instruction format.
- Also, the memory operands appear only in load or store instruction type in MIPS. So that the execute stage can calculate the memory address and then access memory in the following stage.
- Operands must be aligned in memory. Hence, a single data transfer instruction requiring two data memory accesses can be done in a single pipeline stage.

Hazards in Pipelining

Hazards are situations that prevent the next instruction in the instruction cycle from being executing during its designated clock cycle. Hazards reduce the performance of the pipelining.

They attempt to use same resource by two or more instructions at the same time.

Example: In case of single memory is used for instructions and data access and when two instructions are accessing the same register one at instruction fetch stage and other at memory access stage. This leads to inconsistent data access.

Types of hazard:

1. **Structural Hazards:** They arise from resource conflicts when the hardware cannot support all possible combinations of instructions in simultaneous overlapped execution.

2. **Data Hazards:** They arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of instructions in the pipeline.
3. **Control Hazards:** They arise from the pipelining of branches and other instructions that change the PC. This is also known as **branch hazard**. The flow of instruction addresses is not what the pipeline had expected. This results in control hazard.

Data Hazards

Data hazards occur when the pipeline must be stalled because one step must wait for another to complete.

Data hazards occur in register files due to inconsistencies in file. This is an occurrence in which a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction is not yet available. In other words, data hazards occur when the pipeline must be stalled because one step must wait for another to complete. This is due to the data dependence.

Example : Consider the following instructions:

add \$s0, \$t0, \$t1

sub \$t2, \$s0, \$t3

Here the sub instruction uses the result of add instruction (\$s0). The add instruction cannot not write its result until the fifth stage. This results in wasting three clock cycles in the pipeline. Since the stall occurs due to the non availability of data, this is termed as data hazards.

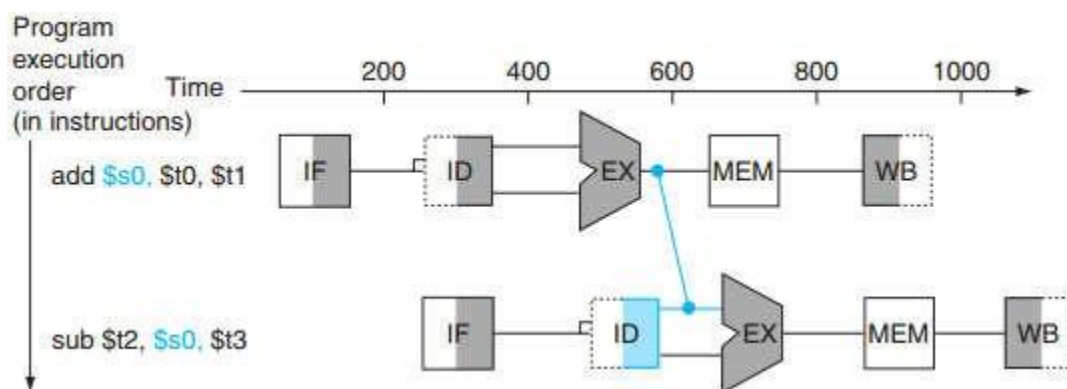


Fig 3.15: Data Hazard

Solution to resolve data hazard:

Forwarding or bypassing is a method of resolving a data hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmable registers or memory. This can be done by adding extra memory element or hardware that acts as an internal buffer.

Forwarding cannot be a universal solution to solve data hazards. Consider the following instructions:

```
lw $s0, 20($t1)
```

```
sub $t2, $s0, $t3
```

The desired data would be available only after the fourth stage of the first instruction in the dependence, which is too late for the input of the third stage of sub. Hence, even with forwarding, there will be a hazard called as **load-use data hazard**.

A specific form of data hazard in which the data requested by a load instruction has not yet become available when it is requested. This is Load-use data hazard.

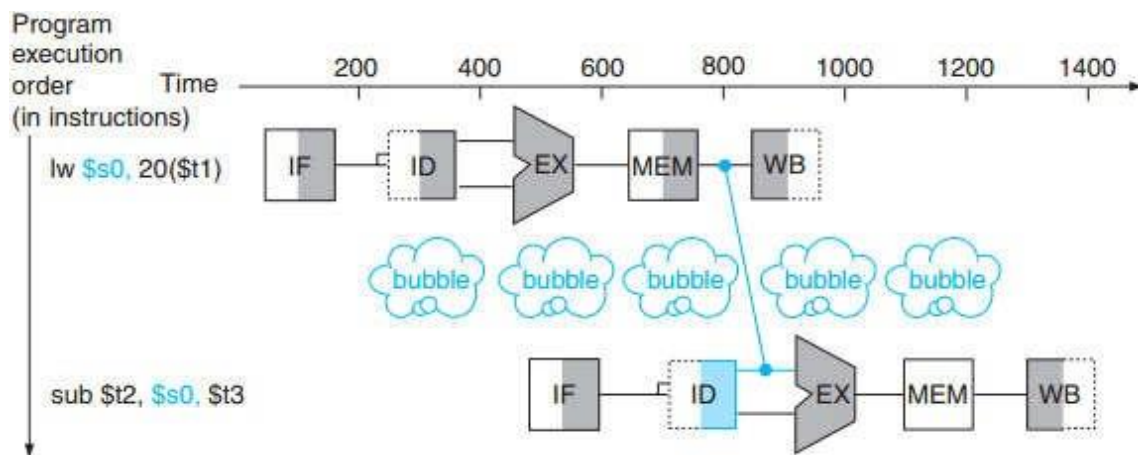


Fig 3.16: Load-Use data hazard

The stall mentioned in Fig 3.16 is called **bubble** or **pipeline stall**. A pipeline stall is a delay in execution of an instruction in order to resolve a hazard. During the decoding stage, the control unit will determine if the decoded instruction reads from a register that the instruction currently in the execution stage writes to.

Problem 3.1

Find the hazards in the following code segment and reorder the instructions to avoid any pipeline stalls.

```
lw $t1, 0($t0)
lw $t2, 4($t0)
add $t3, $t1,$t2
sw $t3, 12($t0)
lw $t4, 8($t0)
add $t5, $t1,$t4
sw $t5, 16($t0)
```

Solution :

Both the add instructions have a hazard because of their dependence on the immediately preceding lw instruction. Bypassing eliminates several other potential hazards including the dependence of the first add on the first lw and any hazards for store instructions. Moving up the third lw instruction eliminates both hazards. This is possible since the lw instruction is independent of other operations:

```
lw $t1, 0($t0)
lw $t2, 4($t1)
lw $t4, 8($t0)
add $t3, $t1,$t2
sw $t3, 12($t0)
add $t5, $t1,$t4
sw $t5, 16($t0)
```

A PIPELINED DATAPATH

The stages of pipelined datapath are:

1. IF: Instruction fetch
2. ID: Instruction decode and register file read

3. EX: Execution or address calculation
4. MEM: Data memory access
5. WB: Write back

The two exceptions to the normal flow of instructions:

1. The write-back stage, which places the result back into the register file in the middle of the datapath.
2. The selection of the next value of the PC, choosing between the incremented PC and the branch address from the MEM stage.

Data flowing from right to left does not affect the current instruction; only later instructions in the pipeline are influenced by these reverse data movements. Note that the first right-to-left arrow can lead to data hazards and the second leads to control hazards. One way to show what happens in pipelined execution is to pretend that each instruction has its own datapath, and then to place these datapaths on a time line to show their relationship.

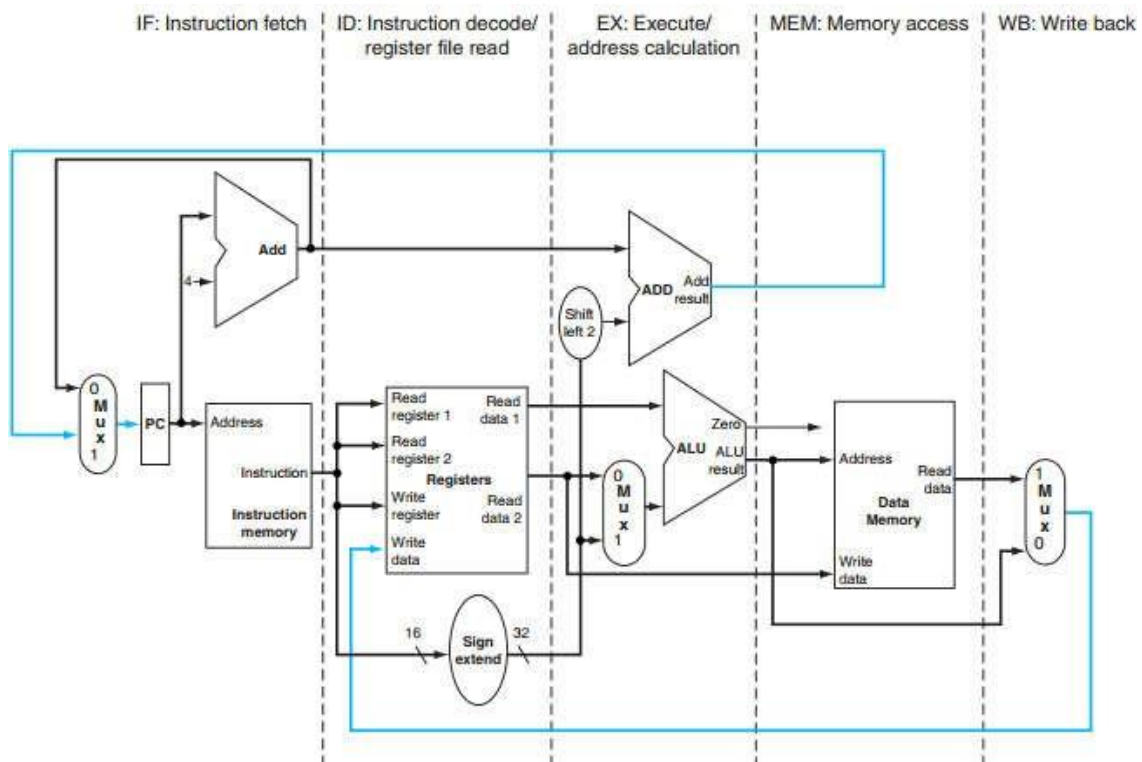


Fig 3.17: Single cycle datapath

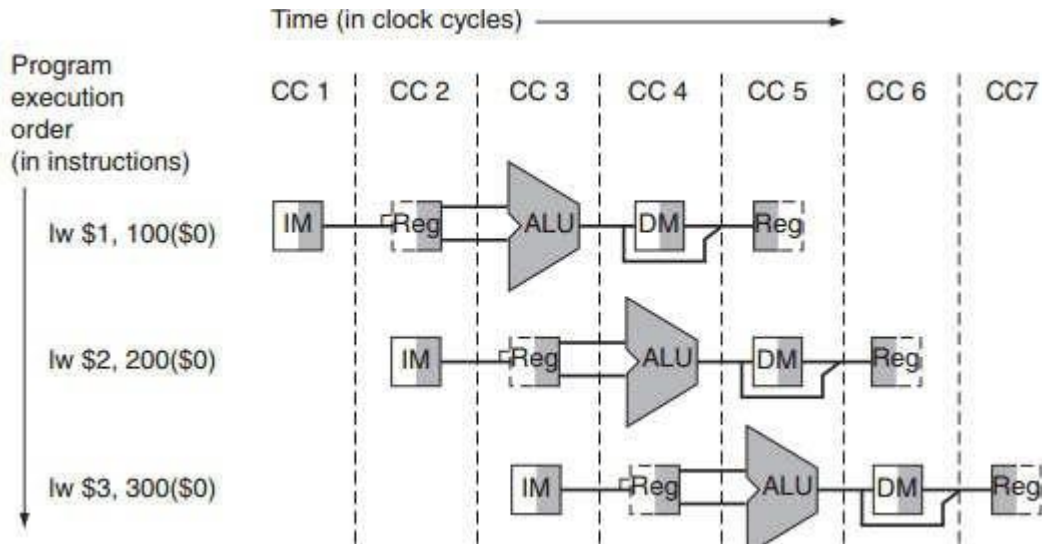


Fig 3.18: Instructions in single cycle datapath

- The above fig shows that each instruction has its own datapath, and each stage is labeled by the physical resource used in that stage, corresponding to the portions of the datapath.
- IM represents the instruction memory and the PC in the instruction fetch stage, Reg stands for the register file and sign extender in the instruction decode/register file read stage (ID), and so on.
- To maintain proper time order, the datapath breaks the register file into two logical parts: registers read during register fetch (ID) and registers written during write back (WB).
- This dual use is represented by drawing the unshaded left half of the register file using dashed lines in the ID stage, when it is not being written, and the unshaded right half in dashed lines in the WB stage, when it is not being read.
- As before, we assume the register file is written in the first half of the clock cycle and the register file is read during the second half.

Operations in each stage of Pipeline:

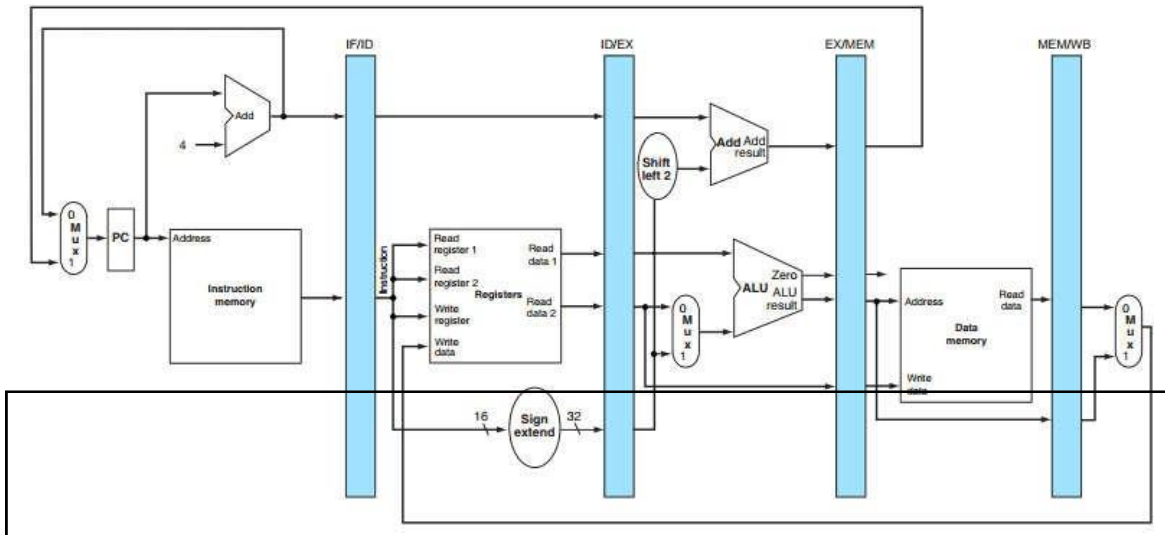


Fig 3.19: Five stages of Pipeline

1. Instruction fetch:

- The instruction is read from memory using the address in the PC and then placed in the IF/ID pipeline register.
- The IF/ID pipeline register is similar to the Instruction register. The PC address is incremented by 4 and then written back into the PC to be ready for the next clock cycle.
- This incremented address is also saved in the IF/ID pipeline register in case it is needed later for an instruction, such as beq.
- The computer cannot know which type of instruction is being fetched, so it must prepare for any instruction, passing potentially needed information down the pipeline.

2. Instruction decode and register file read:

- The instruction portion of the IF/ID pipeline register supplying the 16-bit immediate field, which is sign-extended to 32 bits, and the register numbers to read the two registers.

- All three values are stored in the ID/EX pipeline register, along with the incremented PC address.
- Transfer everything that might be needed by any instruction during a later clock cycle.
- These first two stages are executed by all instructions, since it is too early to know the type of the instruction.

3. Execute or address calculation:

- The load instruction reads the contents of register 1 and the sign-extended immediate from the ID/EX pipeline register and adds them using the ALU.
- That sum is placed in the EX/MEM pipeline register.

4. Memory access:

- The load instruction reading the data memory using the address from the EX/MEM pipeline register and loading the data into the MEM/WB pipeline register.
- The register containing the data to be stored was read in an earlier stage and stored in ID/EX.
- The only way to make the data available during the MEM stage is to place the data into the EX/MEM pipeline register in the EX stage, just as we stored the effective address into EX/MEM.

5. Write back:

- This involves reading the data from the MEM/WB pipeline register and writing it into the register file.

PIPELINED CONTROL

This section describes the necessary control lines for implementing a pipelined datapath. The control logic is needed for PC source, register destination number, and ALU control. A 6-bit funct field (function code) is needed for the instruction in the EX stage as input to ALU control, so these bits must also be included in the ID/EX pipeline register. These 6 bits are the 6 least significant bits of the immediate field in the instruction, so the ID/EX pipeline register can supply them from the immediate field since sign extension leaves these bits unchanged.

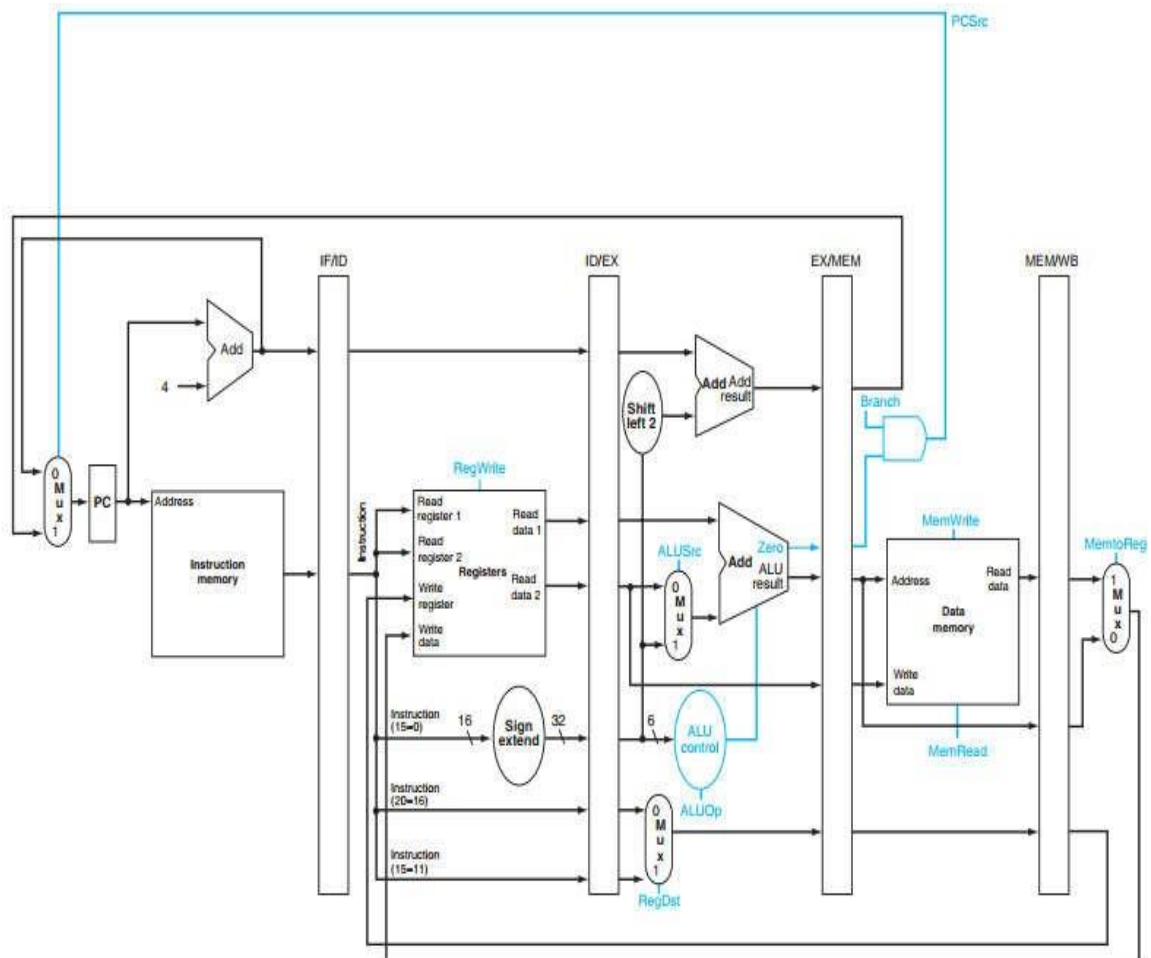


Fig 3.20: Control signals in single cycled data path

Sequence of operations:

- The PC is written on each clock cycle, so there is no separate write signal for the PC.
- There are no separate write signals for the pipeline registers (IF/ID, ID/EX, EX/MEM, and MEM/WB), since the pipeline registers are also written during each clock cycle.
- To specify control for the pipeline, set the control values during each pipeline stage. Because each control line is associated with a component active in only a single pipeline stage.
- The control lines are also divided into five groups according to the pipeline stage:

1. **Instruction fetch:** The control signals to read instruction memory and to write the PC are always asserted, so there is nothing special to control in this pipeline stage.
2. **Instruction decode/register file read:** As in the previous stage, the same thing happens at every clock cycle, so there are no optional control lines to set.
3. **Execution/address calculation:** The signals to be set are RegDst, ALUOp, and ALUSrc. The signals select the Result register, the ALU operation, and either Read data 2 or a sign-extended immediate for the ALU.
4. **Memory access:** The control lines set in this stage are Branch, MemRead, and MemWrite. These signals are set by the branch equal, load, and store instructions, respectively.
5. **Write back:** The two control lines are MemtoReg, which decides between sending the ALU result or the memory value to the register file, and RegWrite, which writes the chosen value.

Implementing control means setting the nine control lines to these values in each stage for each instruction (explained in simple implementation scheme). The simplest way to do this is to extend the pipeline registers to include control information.

DATA HAZARDS

Data hazards occur when the pipeline must be stalled because one step must wait for another to complete.

Data hazards occur in register files due to inconsistencies in file. This is an occurrence in which a planned instruction cannot execute in the proper clock cycle because data that is needed to execute the instruction is not yet available. In other words, data hazards occur when the pipeline must be stalled because one step must wait for another to complete. This is due to the data dependence.

Forwarding or Bypassing

Forwarding or bypassing is a method of resolving a data hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmer visible registers or memory. This can be done by adding extra memory element or hardware that acts as an internal buffer.

Forwarding cannot be a universal solution to solve data hazards. Consider the following instructions:

```
lw $s0, 20($t1)
```

```
sub $t2, $s0, $t3
```

The desired data would be available only after the fourth stage of the first instruction in the dependence, which is too late for the input of the third stage of sub. Hence, even with forwarding, there will be a hazard called as **load-use data hazard**.

A specific form of data hazard in which the data requested by a load instruction has not yet become available when it is requested. This is Load-use data hazard.

Consider the following code:

```
sub $2, $1, $3
```

```
and $12, $2, $5
```

```
or $13, $6, $2
```

```
add $14, $2, $2
```

```
sw $15, 100($2)
```

There are several dependences in this code fragment:

- The first instruction, SUB, stores a value into \$2.
- That register is used as a source in the rest of the instructions. This is no problem for 1-cycle and multicycledatapath.
- Each instruction executes completely before the next begins.
- This ensures that instructions 2 through 5 above use the new value of \$2.

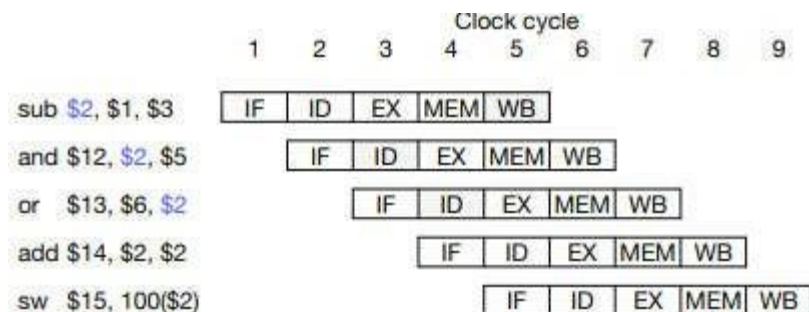


Fig 3.21: Pipelined diagram

- The SUB does not write to register \$2 until clock cycle 5 causing 2 data hazards in our pipelined datapath.
- The AND reads register \$2 in cycle 3. Since SUB hasn't modified the register yet, this is the old value of \$2
- The OR instruction uses register \$2 in cycle 4, again before it's actually updated by SUB.

To avoid data hazard, rewrite the instructions (sll means stall):

```
sub $2, $1, $3
sll $0, $0, $0
sll $0, $0, $0
and $12, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)
```

Since it takes two instruction cycles to get the value stored, one solution is for the assembler to insert no-ops or for compilers to reorder instructions to do useful work while the pipeline proceeds. Since the pipeline registers already contain the ALU result, we could just forward the value to later instructions, to prevent data hazards

- In clock cycle 4, the AND instruction can get the value of \$1 - \$3 from the EX/MEM pipeline register used by SUB.
- Then in cycle 5, the OR can get that same result from the MEM/WB pipeline register being used by SUB.

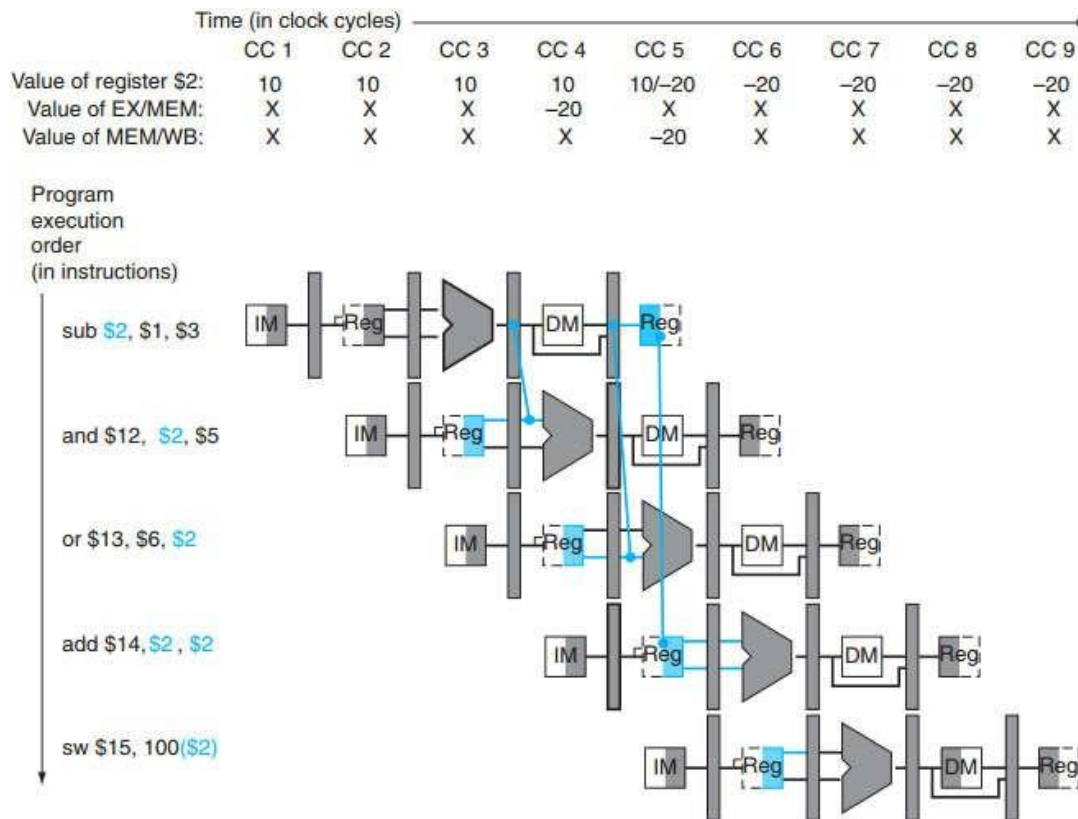


Fig 3.22: Pipelined dependencies

Forward the data as soon as it is available to any units that need it before it is available to read from the register file. This is forwarding in data hazards.

When an instruction tries to use a register in its EX stage that an earlier instruction intends to write in its WB stage, we actually need the values as inputs to the ALU. The general format for specifying dependencies is given by:

Pipeline register. Field in the register

Example: ID/EX.RegisterRs- refers that the value is found in the pipeline register ID/EX in the field RegisterRs. The dependencies in the given example are:

- EX/MEM.RegisterRd = ID/EX.RegisterRs
- EX/MEM.RegisterRd = ID/EX.RegisterRt
- MEM/WB.RegisterRd = ID/EX.RegisterRs
- MEM/WB.RegisterRd = ID/EX.RegisterRt

The first hazard in the sequence is on register \$2, between the result of sub \$2,\$1,\$3 and the first read operand of and \$12,\$2,\$5. This hazard can be detected when the and instruction is in the EX stage and the prior instruction is in the MEM stage.

$$EX/MEM.RegisterRd = ID/EX.RegisterRs = \$2$$

Forwarding the inputs to the ALU from any pipeline registers done by adding multiplexors to the input of the ALU and with the proper controls. By this the pipeline can be executed at full speed in the presence of these data dependences.

Stalling

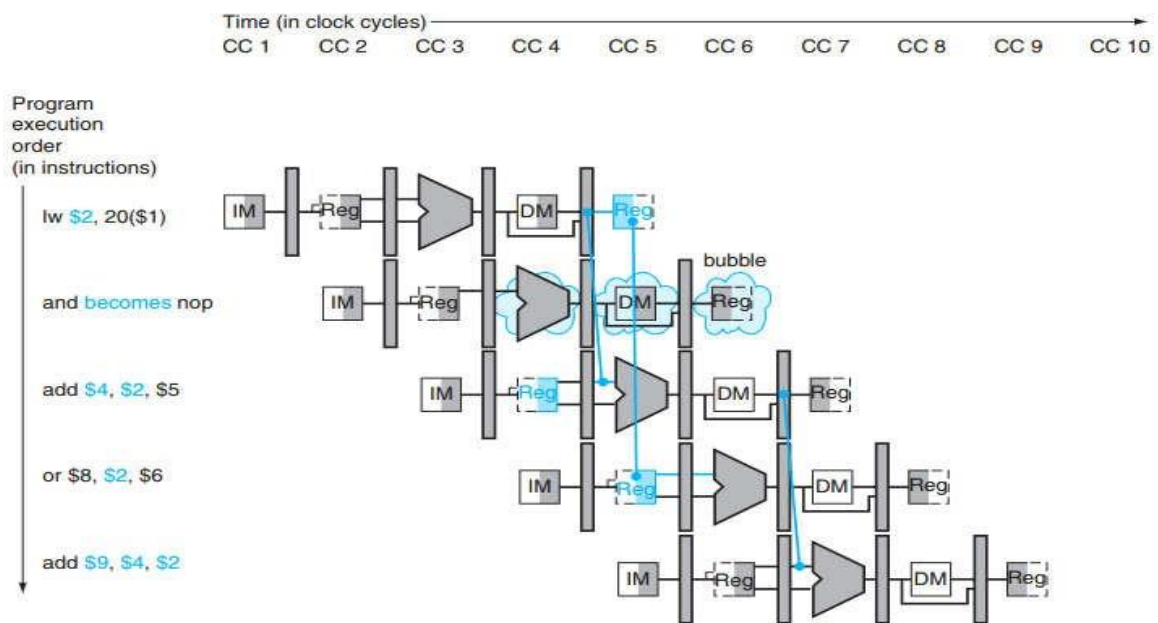


Fig 3.23: Introducing stalls in pipelining

A bubble is inserted beginning in clock cycle 4, by changing the and instruction to a nop (no operation). Note that the and instruction is really fetched and decoded in clock cycles 2 and 3, but its EX stage is delayed until clock cycle 5. The or instruction is fetched in clock cycle 3, but its IF stage is delayed until clock cycle 5. After insertion of the bubble, all the dependences go forward in time and no further hazards occur.

In short forwarding requires:

- (a) Recognizing when a potential data hazard exists, and
- (b) Revising the pipeline to introduce forwarding paths.

CONTROL HAZARDS

This occurs when there is a need for an instruction to take a decision based on the results of another instruction's result that has not yet completed its execution.

Control or branching hazards arise from resource conflicts when the hardware cannot support all possible combinations of instructions in simultaneous overlapped execution.

Instructions that disrupt the sequential flow of control present problems for pipelines and are potential candidates for control hazards. The effects of these instructions cannot be exactly determined until late in the pipeline, so instruction fetch cannot continue unless it is explicitly managed. The following types of instructions can introduce control hazards:

- Unconditional branches
- Conditional branches
- Indirect branches
- Procedure calls
- Procedure returns

Example:

```

ld    r2, 0(r4)    // r2 := memory at r4
ld    r3, 4(r4)    // r3 := memory at r4+4
sub   r1, r2, r3   // r1 := r2 - r3
beqz  r1, L1       // if r1 is not 0, goto L1
ldi   r1, 1        // r1 := 1
L1:   not   r1, r1  // r1 := not r1
st    r1, 0(r5)    // store r1 to memory at r5

```

This code compares two memory locations and stores the result of that comparison (1 for equal, 0 for not equal) to another location. If the beqz branch is taken, then a 1 is stored; otherwise, a 0 is stored. The beqz instruction sources two hazards:

1. When the beqz instruction is in the decode stage, the sub instruction is in the execute stage. The branch cannot read the output of the sub until it has been written to the register file; if it reads it early, it will read the wrong value.
2. The instruction that is to be fetched after beqz is not known in advance. At this point, the status of the branch instruction is totally unknown whether it depends on the previous instruction or not. This is because it hasn't been decoded yet, so bypassing also can't help in resolving the hazard. Even if the decision is known, the location from where to fetch the instruction if the branch is taken is unknown because the effective address computation for branches do not happen until the EX stage.

Solutions for control hazards:

The following are solutions that can reduce control hazards:

1. **Pipeline stall cycles:** Freeze the pipeline until the branch outcome and target are known, then proceed with fetch. Thus, every branch instruction incurs a penalty equal to the number of stall cycles. This solution is unsatisfactory if the instruction mix contains many branch instructions, and/or the pipeline is very deep.
2. **Branch delay slots:** The instruction set architecture is constructed such that one or more instructions sequentially following a conditional branch instruction are executed whether or not the branch is taken. The compiler or assembly language writer must fill these branch delay slots with useful instructions or NOPs (no-operation opcodes).
3. **Branch prediction:** The outcome and target of conditional branches are predicted using some heuristic. Instructions are speculatively fetched and executed down the predicted path, but results are not written back to the register file until the branch is executed and the prediction is verified. When a branch is predicted, the processor enters a speculative mode in which results are written to another register file that mirrors the architected register file. Another pipeline stage called the commit stage is introduced to handle writing verified speculatively obtained results back into the real register file. Branch predictors can't be 100% accurate, so there is still a penalty for branches that is based on the branch misprediction rate.

4. **Indirect branch prediction:** Branches such as virtual method calls, computed gotos and jumps through tables of pointers can be predicted using various techniques.
5. **Return address stack (RAS):** Procedure returns are a form of indirect jump that can be perfectly predicted with a stack as long as the call depth doesn't exceed the stack depth. Return addresses are pushed onto the stack at a call and popped off at a return.

Static Branch Prediction

Branch prediction is a method of resolving a branch hazard that assumes a given outcome for the branch and proceeds from that assumption rather than waiting for the actual outcome.

- In general, the bottoms of loops are branches that jump back to the top of the loop. These types of loops can easily be predicted as branch taken.
- The decision about a branch whether taken or not taken is arrived from the heuristics.
- **Dynamic hardware predictors**, guess the behaviour of each branch and may change predictions for a branch over the life of a program.
- Dynamic prediction is performed by maintaining a history for each branch as taken or untaken, and then using the recent past behavior to predict the future.
- When the guess is wrong, the pipeline control must ensure that the instructions following the wrongly guessed branch have no effect and must restart the pipeline from the proper branch address.

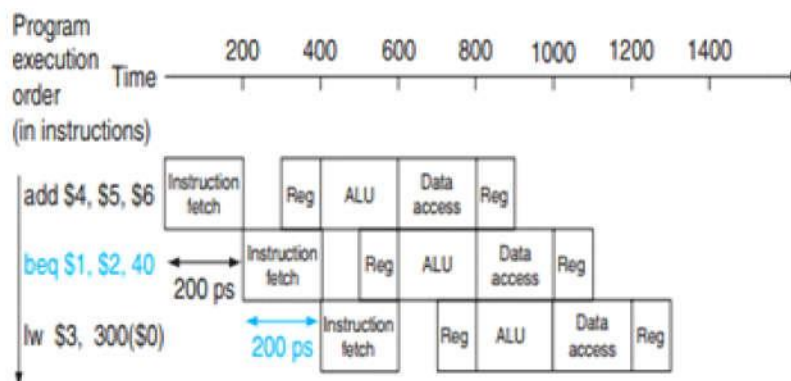


Fig 3.24 a) Branch not taken

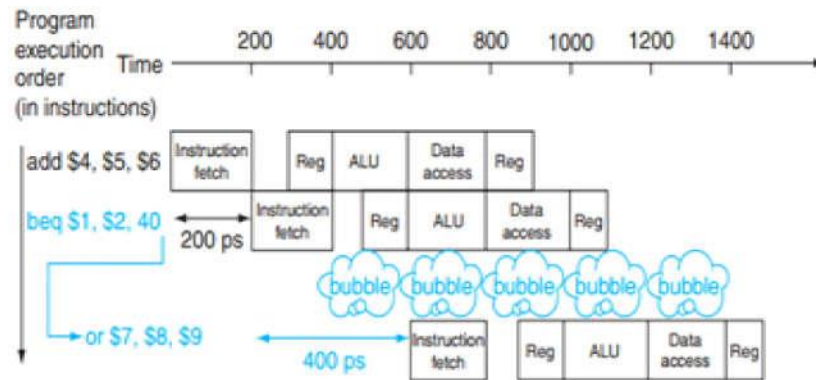


Fig 3.24 b) Branch taken

Branch Stalling

- This is stalling the instructions until the branch is complete is too slow.
- One improvement over branch stalling is to predict that the branch will not be taken and thus continue execution down the sequential instruction stream.
- If the branch is taken, the instructions that are being fetched and decoded must be discarded. Execution continues at the branch target.
- If branches are untaken half the time, and if it costs little to discard the instructions, this optimization halves the cost of control hazards.
- To discard instructions, change the original control values to 0s.

Delayed Branches:

The delayed branch always executes the next sequential instruction, with the branch taking place after that one instruction delay. It is hidden from the MIPS assembly language programmer because the assembler can automatically arrange the instructions to get the branch behaviour desired by the programmer.

- One way to improve branch performance is to reduce the cost of the taken branch.
- The MIPS architecture was designed to support fast single-cycle branches that could be pipelined with a small branch penalty.

- Moving the branch decision up requires two actions to occur earlier:
 1. Computing the branch target address
 2. Evaluating the branch decision.
- The easy part of this change is to move up the branch address calculation.
- Despite these difficulties, moving the branch execution to the ID stage is an improvement, because it reduces the penalty of a branch to only one instruction if the branch is taken, namely, the one currently being fetched.

Dynamic Branch Prediction

Prediction of branches at runtime using runtime information is called dynamic branch prediction.

- One implementation of that approach is a branch prediction buffer or branch history table.
- A **branch prediction buffer** is a small memory indexed by the lower portion of the address of the branch instruction.
- The memory contains a bit that says whether the branch was recently taken or not.

1 bit Prediction scheme

This scheme will be incorrect twice when not taken:

- Assume `predict_bit=0` to start (indicates branch not taken) and loop control is at the bottom of the code.
- First iteration in the loop, the predictor mispredicts the branch since the branch is taken back to the top of the loop. Now invert the prediction bit (`predict_bit=1`).
- Till the branch is taken, the prediction is correct.
- Exiting the loop, the predictor again mispredicts the branch since this time the branch is not taken falling out of the loop. Now invert the prediction bit (**`predict_bit=0`**).

Loop: first loop instruction

Second loop instruction

-

-

-

Last loop instruction

Bne \$1,\$2, loop

Fall out instruction

2 bit prediction scheme:

By using 2 bits rather than 1, a branch that strongly favors taken or not taken—as many branches do—will be mispredicted only once. The 2 bits are used to encode the four states in the system. The two-bit scheme is a general instance of a counter-based predictor, which is incremented when the prediction is accurate and decremented otherwise, and uses the midpoint of its range as the division between taken and not taken.

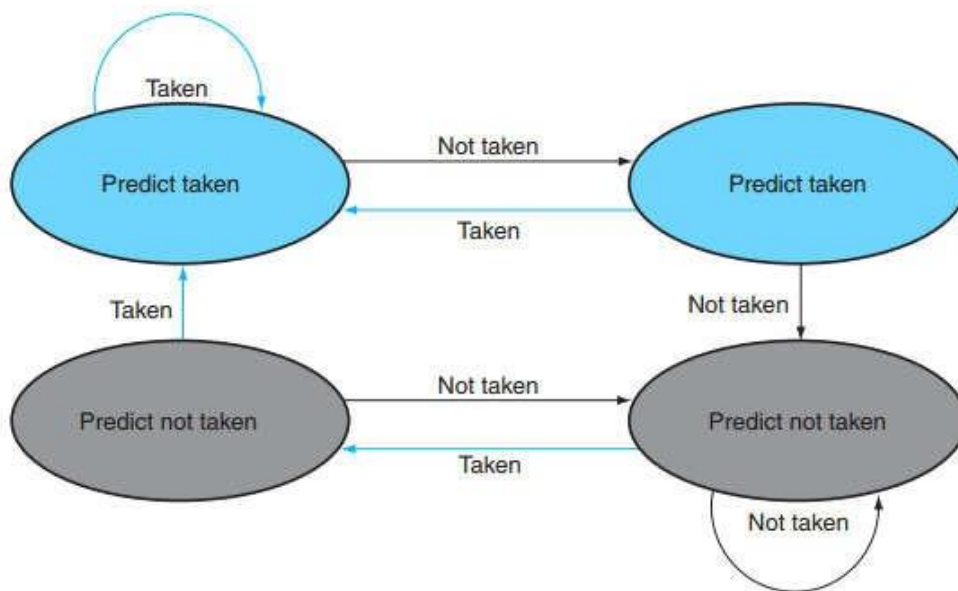


Fig 3.24: 2 bit prediction scheme

Branch delay slot:

- The slot directly after a delayed branch instruction, which in the MIPS architecture is filled by an instruction that does not affect the branch.

- The limitations on delayed branch scheduling arise from
 - (1) the restrictions on the instructions that are scheduled into the delay slots
 - (2) the ability to predict at compile time whether a branch is likely to be taken or not.
- Delayed branching was a simple and effective solution for a five-stage pipeline issuing one instruction each clock cycle.
- As processors go to both longer pipelines and issuing multiple instructions per clock cycle, the branch delay becomes longer, and a single delay slot is insufficient.
- Hence, delayed branching has lost popularity compared to more expensive but more flexible dynamic approaches.

EXCEPTIONS

Control is the most challenging aspect of processor design: One of the hardest parts of control is implementing exceptions and interrupts events other than branches or jumps that change the normal flow of instruction execution. They were initially created to handle unexpected events from within the processor, like arithmetic overflow. The term exception refers to any unexpected change in control flow without distinguishing whether the cause is internal or external. Interrupt is when the event is externally caused. The following are the causes of exceptions:

- R-type arithmetic overflow
- Executing undefined instruction
- I/O device request
- OS service request
- Hardware malfunction

Event	Location	MIPS term
I/O device request	External	Interrupt
OS service request	Internal	Exception
R-type arithmetic overflow	Internal	Exception
Executing undefined instruction	Internal	Exception
Hardware malfunction	Either	Exception / Interrupt

Detecting exceptional conditions and taking the appropriate action is often on the critical timing path of a processor, which determines the clock cycle time and performance.

Exception Handling in the MIPS Architecture:

- The two types of exceptions that MIPS implementation can generate are execution of an undefined instruction and an arithmetic overflow.
- The basic action that the processor must perform when an exception occurs is to save the address of the off ending instruction in the **exception program counter (EPC)** and then transfer control to the operating system at some specified address.
- The operating system can then take the appropriate action, which may involve providing some service to the user program, taking some predefined action in response to an overflow, or stopping the execution of the program and reporting an error.
- After performing whatever action is required because of the exception, the operating system can terminate the program or may continue its execution, using the EPC to determine where to restart the execution of the program.
- For the operating system to handle the exception, it must know the reason for the exception, in addition to the instruction that caused it.
- There are two main methods used to communicate the reason for an exception.
 1. The method used in the MIPS architecture is to include a status register :**the Cause register that** holds a field indicating the reason of exception.
 2. Use **vectored interrupts**. In a vectored interrupt, the address to which control is transferred is determined by the cause of the exception. This demands the inclusion of two external registers :

EPC: A 32-bit register used to hold the address of the affected instruction.

Cause: A register used to record the cause of the exception. In the MIPS architecture, this register is 32 bits, although some bits are currently unused.

Exceptions in a Pipelined Implementation:

A pipelined implementation treats exceptions as another form of control hazard. For example, suppose there is an arithmetic overflow in an add instruction. Flush the instructions that follow the add instruction from the pipeline and begin fetching instructions

from the new address. This is done by turning the IF stage into a nop. Because of careful planning, the overflow exception is detected during the EX stage; hence, we can use the EX.Flush signal to prevent the instruction in the EX stage from writing its result in the WB stage. The final step is to save the address of the off ending instruction in the exception program counter (EPC). In reality, we save the address +4, so the exception handling the software routine must first subtract 4 from the saved value.

PARALLELISM VIA INSTRUCTIONS

The simultaneous execution of multiple instructions from a program is called Instruction Level Parallelism (ILP). It is a measure of how many of the instructions in a computer program can be executed simultaneously.

In Multiple Issue technique, multiple instructions are launched in one clock cycle.

The ILP increases the depth of the pipeline to overlap more instructions. This is facilitated by adding extra hardware resources to replicate the internal component of the computer, so that it can launch multiple instructions in every pipeline stages. This is called **multiple issue**.

This will improve the performance of the processor. The pipelined performance is estimated from the given formula (CPI-Cycles Per Instruction):

$$\text{Pipeline CPI} = \text{Ideal CPI} + \text{Structural stalls} + \text{RAW stalls} + \text{WAR stalls} + \text{WAW stalls} + \text{Control stall}$$

Launching multiple instructions per stage allows the instruction execution rate (CPI) to be less than 1. To obtain substantial increase in performance, we need to exploit parallelism across multiple basic blocks.

Implementing multiple issue processor

- **Static multiple issue processor:** Here the decisions are made by the compiler before execution.
- **Dynamic multiple issue processor:** Here the decisions are made during the execution by the processor.

The challenges in implementing a multiple issue pipeline are:

- **Packaging instructions into issue slots:** Issue slots are the positions from which instructions could be issued in a given clock cycle. To find the exact location of the current issue slot is the greatest challenge. So the process is partially handled by the compiler. ; In dynamic issue designs, it is normally dealt with at runtime by the processor.
- **Dealing with data and control hazards:** In static issue processors, the consequences of data and control hazards are handled statically by the compiler. In dynamic issue processors, use hardware techniques to mitigate the control and data hazard.

Speculation

Speculation is an approach whereby the compiler or processor guesses the outcome of an instruction to remove it as a dependence in executing other instructions.

- This allows the execution of complete instructions or parts of instructions before being certain whether this execution should take place.
- A commonly used form of speculative execution is control flow speculation where instructions past a control flow instruction are executed before the target of the control flow instruction is determined.
- Speculation may be done in the compiler or by the hardware.
- The uses speculation to reorder instructions, moving an instruction across a branch or a load across a store. The compiler usually inserts additional instructions that check the accuracy of the speculation and provide a fix-up routine to use when the speculation was incorrect.
- The processor hardware can perform the same transformation at runtime using techniques. The processor usually buffers the speculative results until it knows they are no longer speculative. If the speculation was correct, the instructions are completed by allowing the contents of the buffers to be written to the registers or memory. If the speculation was incorrect, the hardware flushes the buffers and re-executes the correct instruction sequence.

Issue in Speculation:

Speculating on certain instructions may **introduce exceptions** that were formerly not present. The result would be that an exception that should not have occurred will occur. In

compiler-based speculation, such problems are avoided by adding special speculation support that allows such exceptions to be ignored until it is clear that they really should occur. In hardware-based speculation, exceptions are simply buffered until it is clear that the instruction causing them is no longer speculative and is ready to complete; at that point the exception is raised, and normal exception handling proceeds.

Static Multiple Issue

The set of instructions that issues together in 1 clock cycle; the packet may be determined statically by the compiler or dynamically by the processor.

Static multiple-issue processors use compiler to assist with packaging instructions and handling hazards. The issue packet is treated as one large instruction with multiple operations. This is otherwise termed as **Very Long Instruction Word (VLIW)**. Since the Intel IA-64 architecture supports this approach, it is known as **Explicitly Parallel Instruction Computer (EPIC)**.

Loop unrolling is a technique used by compiler to solve static multiple issue.

Loop Unrolling is a technique to get more performance from loops that access arrays, in which multiple copies of the loop body are made and instructions from different iterations are scheduled together.

Loop unrolling is a compiler optimization applied to certain kinds of loops to reduce the frequency of branches and loop maintenance instructions. It is easily applied to sequential array processing loops where the number of iterations is known prior to execution of the loop. After unrolling, there is more ILP available by overlapping instructions from different iterations.

- During the unrolling process, the compiler introduced additional registers, since multiple copies of the loop body are made.
- Augmenting new registers in loop unrolling is called **register renaming**. This is done to eliminate dependences that are not true data dependences, but may lead to potential hazards or may prevent the compiler from scheduling the code.
- To identify the independent instructions, it is necessary to trace the data dependencies.
- If there is no data values flow between the instructions, it is termed as **anti-dependence or name dependence**. This is an ordering forced purely by the reuse of a name.

- Renaming the registers during the unrolling process allows the compiler to the independent instructions for better code schedule.
- An **instruction group** is a sequence of consecutive instructions with no register data dependences among them.
- All the instructions in a group could be executed in parallel if sufficient hardware resources existed and if any dependences through memory were preserved.
- The compiler must explicitly indicate the boundary between one instruction group and another. This boundary is indicated by placing a **stop** between two instructions that belong to different groups.
- An explicit indicator of a break between independent and dependent instructions is termed as **stop**.
- **Predication** is a technique that can be used to eliminate branches by making the execution of an instruction dependent on a predicate, rather than dependent on a branch.
- Speculation and Predication improves ILP. Branches reduce the opportunity to exploit ILP by restricting the movement of code.
- Branches within a loop cannot be eliminated by loop unrolling. Predication eliminates this branch, by allowing more flexible exploitation of parallelism.
- **Speculation** consists of separate support for control speculation, which deals with deferring exceptions for speculated instructions, and memory reference speculation, which supports speculation of load instructions.
- Deferred exception handling is supported by adding speculative load instructions, which, when an exception occurs, tag the result as poison.
- **Poison** is the result generated when a speculative load yields an exception, or an instruction uses a poisoned operand. When a poisoned result is used by an instruction, the result is also poison, the software can then check for a poisoned result when it knows that the execution is no longer speculative.
- The speculation on memory references can be made by moving loads earlier than stores on which they may depend. This is done with an advanced load instruction.

- Advanced load is speculative load instruction with support to check for aliases that could invalidate the load. This demands the use of a special table to track the address that the processor loaded from.
- A subsequent instruction must be used to check the status of the entry after the load is no longer speculative.

Dynamic Multiple-Issue Processors

- Dynamic multiple issue processors are implemented using **superscalar processors** that are capable of executing more than one instruction per clock cycle.
- The compiler must schedule the instructions to the processors without any dependencies.
- To facilitate this, **dynamic pipeline scheduling** is performed by providing hardware support for reordering the order of instruction execution so as to avoid stalls.

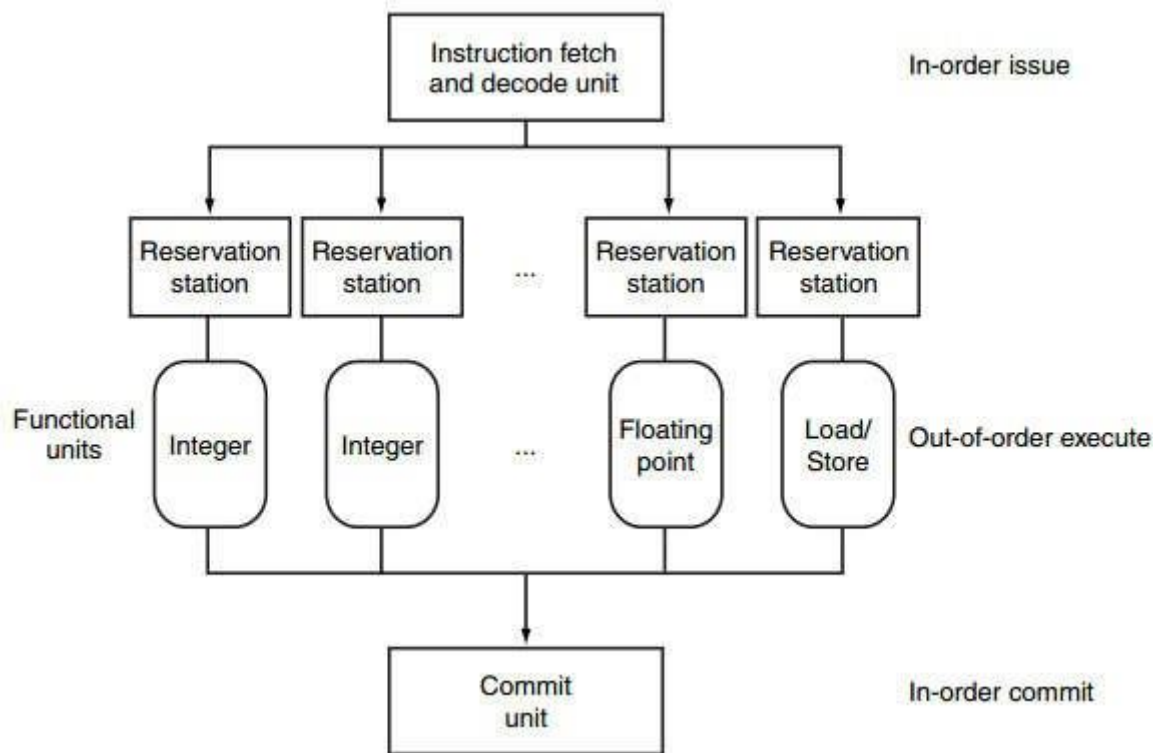


Fig 3.25: Units of dynamic scheduling pipeline

The following are the important components of dynamic scheduling pipelines:

- **Instruction Fetch Unit:** This unit fetches instructions, decodes them, and sends each instruction to a corresponding functional unit for execution.
- **Functional unit:** They have buffers, called **reservation stations**, that hold the operands and the operation. As soon as the buffer contains all its operands and the functional unit is ready to execute, the result is calculated. When the result is completed, it is sent to any reservation stations waiting for this particular result as well as to the commit unit.
- **Commit Unit:** This buffers the result until it is safe to put the result into the register file or, for a store, into memory. The buffer in the commit unit, called the **reorder buffer**, is also used to supply operands, in much the same way as forwarding logic does in a statically scheduled pipeline. Once a result is committed to the register file, it can be fetched directly from there, just as in a normal pipeline.

Operation of dynamic scheduling pipeline:

- When an instruction issues, if either of its operands is in the register file or the reorder buffer, it is copied to the reservation station immediately, where it is buffered until all the operands and an execution unit are available. For the issuing instruction, the register copy of the operand is no longer required, and if a write to that register occurred, the value could be overwritten.
- If an operand is not in the register file or reorder buffer, it must be waiting to be produced by a functional unit. The name of the functional unit that will produce the result is tracked. When that unit eventually produces the result, it is copied directly into the waiting reservation station from the functional unit bypassing the registers.

Dynamic scheduling is often extended by including hardware-based speculation, especially for branch outcomes. By predicting the direction of a branch, a dynamically scheduled processor can continue to fetch and execute instructions along the predicted path.

UNIT - IV

MEMORY AND I/O ORGANIZATION

INTRODUCTION

Memory unit enables us to store data inside the computer. The computer memory always adheres to **principle of locality**.

Principle of locality or locality of reference is the tendency of a processor to access the same set of memory locations repetitively over a short period of time.

Two different types of locality are:

- **Temporal locality:** The principle stating that if a data location is referenced then it will tend to be referenced again soon.
- **Spatial locality:** The locality principle stating that if a data location is referenced, data locations with nearby addresses will tend to be referenced soon.

The locality of reference is useful in implementing the memory hierarchy.

Memory hierarchy is a structure that uses multiple levels of memories; as the distance from the CPU increases, the size of the memories and the access time both increase.

A memory hierarchy consists of multiple levels of memory with different speeds and sizes. The faster memories are more expensive per bit than the slower memories and thus smaller.

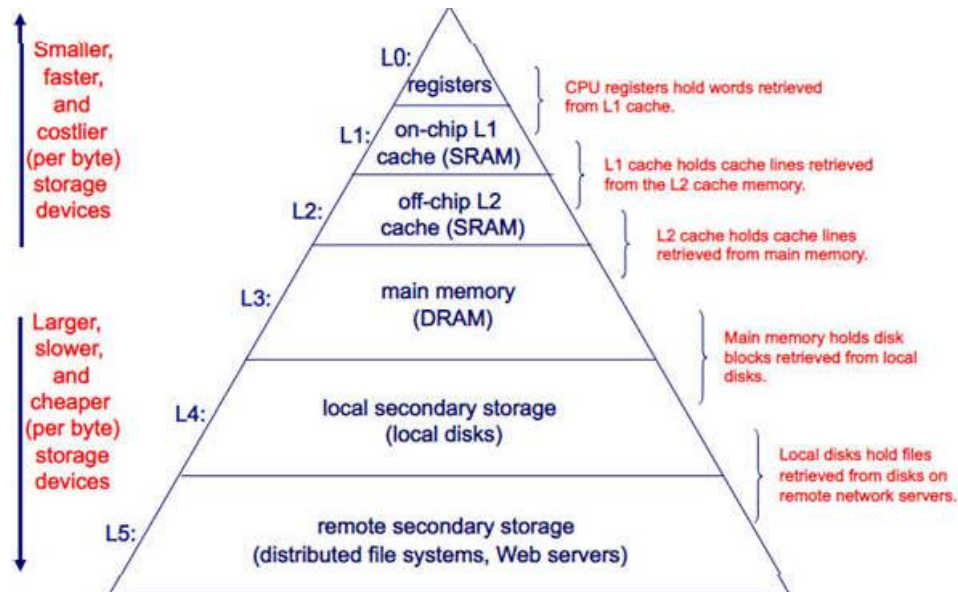


Fig 4.1 Memory Hierarchy

- Main memory is implemented from Dynamic Random Access Memory (DRAM).
- The levels closer to the processor (caches) use Static Random Access Memory (SRAM).
- DRAM is less costly per bit than SRAM, although it is substantially slower.
- For each k , the faster, smaller device at level k serves as a cache for the larger, slower device at level $k+1$.
- The computer programs tend to access the data at level k more often than at level $k+1$.
- The storage at level $k+1$ can be slower

Cache memory (CPU memory) is high-speed SRAM that a computer microprocessor can access more quickly than it can access regular RAM. This memory is typically integrated directly into the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU.

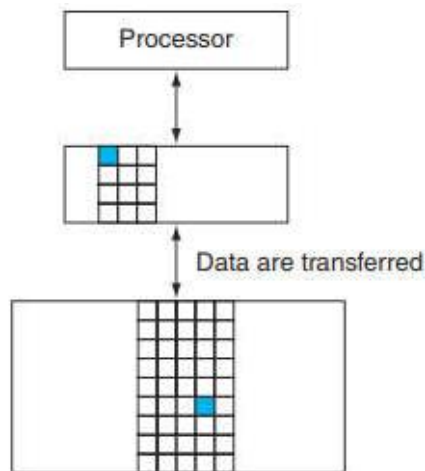


Fig 4.2: Data access by processor

The data transfer between various levels of memory is done through blocks. The minimum unit of information is called a **block**. If the data requested by the processor appears in some block

in the upper level, this is called a **hit**. If the data is not found in the upper level, the request is called a **miss**. The lower level in the hierarchy is then accessed to retrieve the block containing the requested data.

The fraction of memory accesses found in a cache is termed as hit rate or hit ratio.

Miss rate is the fraction of memory accesses not found in a level of the memory hierarchy. Hit time is the time required to access a level of the memory hierarchy, including the time needed to determine whether the access is a hit or a miss.

Miss penalty is the time required to fetch a block into a level of the memory hierarchy from the lower level, including the time to access the block, transmit it from one level to the other, and insert it in the level that experienced the miss.

Because the upper level is smaller and built using faster memory parts, the hit time will be much smaller than the time to access the next level in the hierarchy, which is the major component of the miss penalty.

MEMORY HIERARCHY

A memory unit is a collection of semi-conductor storage cells with circuits to access the data stored in them. The data storage in memory is done in words. The number of bits in a word depends on the architecture of the computer. Generally a word is always multiple of 8. Memory is accessed through unique system assigned address. The accessing of data from memory is based on principle of locality.

Principle of Locality

The locality of reference or the principle of locality is the term applied to situations where the same value or related storage locations are frequently accessed. There are three basic types of locality of reference:

- **Temporal locality:** Here a resource that is referenced at one point in time is referenced again soon afterwards.
- **Spatial locality:** Here the likelihood of referencing a storage location is greater if a storage location near it has been recently referenced.
- **Sequential locality:** Here storage is accessed sequentially, in descending or ascending order. The locality of reference leads to memory hierarchy.

Need for memory hierarchy

Memory hierarchy is an approach for organizing memory and storage systems. It consist of multiple levels of memory with different speeds and sizes. The following are the reasons for such organization:

- Fast storage technologies cost more per byte and have less capacity
- Gap between CPU and main memory speed is widening
- Well-written programs tend to exhibit good locality.

The memory hierarchy is shown in Fig 4.1. The entire memory elements of the computer fall under the following three categories:

- **Processor Memory:**

This is present inside the CPU for high-speed data access. This consists of small set of registers that act as temporary storage. This is the costliest memory component.

- **Primary memory:**

This memory is directly accessed by the CPU. All the data must be brought inside main memory before accessing them. Semiconductor chips acts as main memory.

- **Secondary memory:**

This is cheapest, large and relatively slow memory component. The data from the secondary memory is accessed by the CPU only after it is loaded to main memory.

There is a trade-off among the three key characteristics of memory namely-

- Cost
- Capacity
- Access time

Terminologies in memory access

- **Block or line:** The minimum unit of information that could be either present or totally absent.
- **Hit:** If the requested data is found in the upper levels of memory hierarchy it is called hit.

- **Miss:** If the requested data is not found in the upper levels of memory hierarchy it is called miss.
- **Hit rate or Hit ratio:** It is the fraction of memory access found in the upper level .It is a performance metric.

$$\text{Hit Ratio} = \text{Hit}/(\text{Hit} + \text{Miss})$$

- **Miss rate:** It is the fraction of memory access not found in the upper level (1-hit rate).
- **Hit Time:** The time required for accessing a level of memory hierarchy, including the time needed for finding whether the memory access is a hit or miss.
- **Miss penalty:** The time required for fetching a block into a level of the memory hierarchy from the lower level, including the time to access, transmit, insert it to new level and pass the block to the requestor.
- **Bandwidth:** The data transfer rate by the memory.
- **Latency or access time:** Memory latency is the length of time between the memory's receipt of a read request and its release of data corresponding with the request.
- **Cycle time:** It is the minimum time between requests to memory.

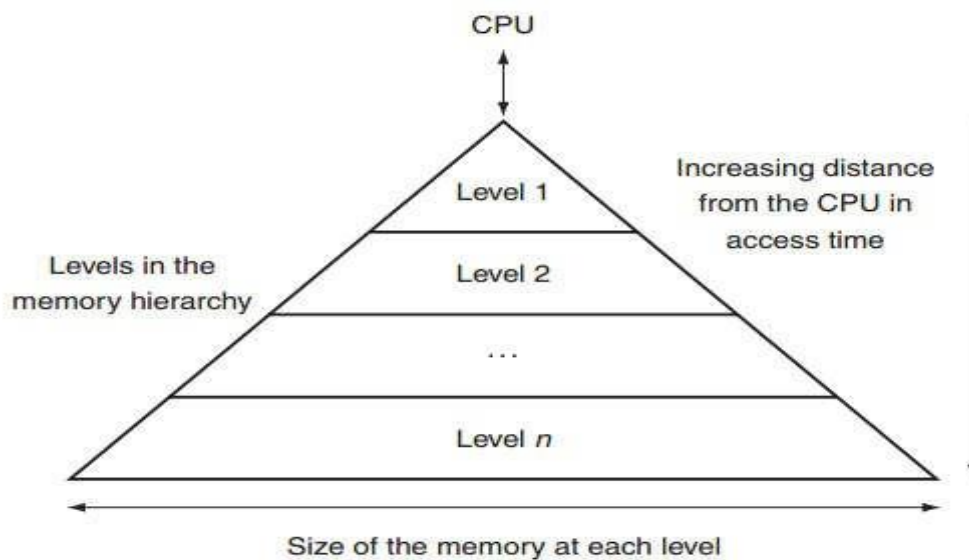


Fig 4.2: Memory level vs Access Time

The memory access time increases as the level increases. Since the CPU registers are located in very close proximity to the CPU they can be accessed very quickly and they are the more costly. As the level increases, the memory access time also increases thereby decreasing the costs.

Levels in Memory Hierarchy

The following are the levels in memory hierarchy:

- **CPU Registers:**

They are at the top most level of this hierarchy, they hold the most frequently used data. They are very limited in number and are the fastest. They are often used by the CPU and the ALU for performing arithmetic and logical operations, for temporary storage of data.

- **Static Random Access Memory (SRAM):**

Static Random Access Memory (Static RAM or SRAM) is a type of RAM that holds data in a static form, that is, as long as the memory has power. SRAM stores a bit of data on four transistors using two cross-coupled inverters. The two stable states characterize 0 and 1. During read and write operations another two access transistors are used to manage the availability to a memory cell.

- **Main memory or Dynamic Random Access Memory (DRAM):**

Dynamic random access memory (DRAM) is a type of memory that is typically used for data or program code that a computer processor needs to function. In other words it is said to be the main memory of the computer. Random access allows processor to access any part of the memory directly rather than having to proceed sequentially from a starting place. The main advantages of DRAM are its simple design, speed and low cost in comparison to alternative types of memory. The main disadvantages of DRAM are volatility and high power consumption relative to other options.

- **Local Disks (Local Secondary Storage):**

A local drive is a computer disk drive that is installed directly within the host or the local computer. It is a computer's native hard disk drive (HDD), which is directly accessed by the computer for storing and retrieving data. It is a cheaper memory with more memory access time.

- **Remote Secondary Storage:**

This includes Distributed file system (DFS) and online storage like cloud. The storage area is vast with low cost but larger access time.

Distinction between Static RAM and Dynamic RAM

SRAM	DRAM
Stores data till the power is supplied.	Stored data only for few milliseconds irrespective of the power supply.
Uses nearly 6 transistors for each memory cell.	Uses single transistor and capacitor for each memory cell.
Do not refresh the memory cell.	Refreshing circuitry is needed.
Faster data access.	Slower access.
Consumes more power.	Low power consumption.
Cost per bit is high.	Comparatively lower costs.
They are made of more number of components per cells.	They are made of less number of components per cells.

CLASSIFICATION OF MEMORY

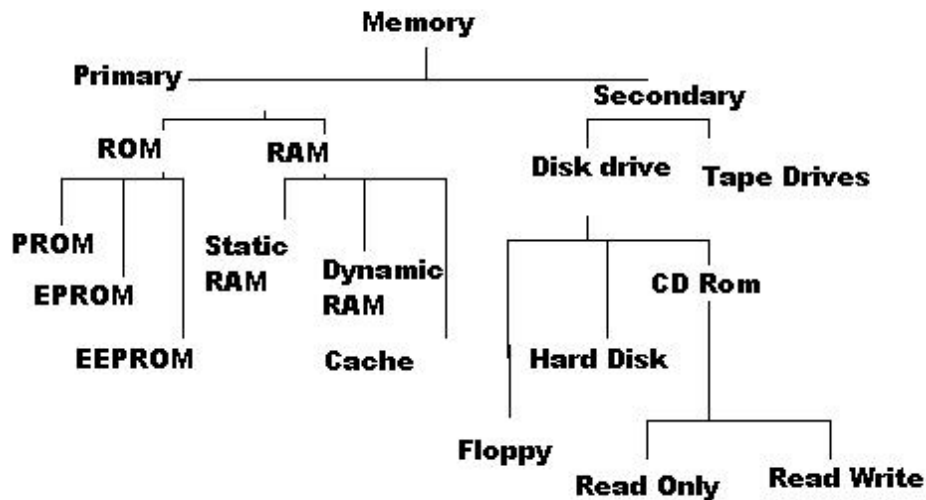


Fig 4.3: Classification of Memory

The instructions and data are stored in memory unit of the computer system are divided into following main groups:

- Main or Primary memory
- Secondary memory.

Primary Memory:

Primary memory is the main area in a computer in which data is stored for quick access by the computer's processor. It is divided into two parts:

i) Random Access Memory (RAM):

RAM is a type of computer primary memory. It accessed any piece of data at any time. RAM stores data for as long as the computer is switched on or is in use. This type of memory is volatile. The two types of RAM are:

- **Static RAM:** This type of RAM is static in nature, as it does not have to be refreshed at regular intervals. Static RAM is made of large number of flip-flops on IC. It is being costlier and having packing density.
- **Dynamic RAM:** This type of RAM holds each bit of data in an individual capacitor in an integrated circuit. It is dynamic in the sense that the capacitor charge is repeatedly refreshed to ensure the data remains intact.

ii) Read Only Memory (ROM):

The ROM is nonvolatile memory. It retains stored data and information if the power is turned off. In ROM, data are stored permanently and can't alter by the programmer. There are four types of ROM:

- **MROM (mask ROM):** MROM (mask ROM) is manufacturer-Programmed ROM in which data is burnt in by the manufacturer of the electronic equipment in which it is used and it is not possible for a user to modify programs or data stored inside the ROM chip.
- **PROM (programmable ROM):** PROM is one in which the user can load and store "read-only" programs and data. In PROM the programs or data are stored only fast time and the stored data cannot modify the user.
- **EPROM (erasable programmable ROM):** EPROM is one in which is possible to erase information stored in an EPROM chip and the chip can be reprogrammed to store new information. When an EPROM is in use, information stored in it can only be "read" and the information remains in the chip until it is erased.
- **EEPROM (electronically erasable and programmable ROM):** EEPROM is one type of EPROM in which the stored information is erased by using high voltage electric pulse. It is easier to alter information stored in an EEPROM chip.

Secondary Memory:

Secondary memory is where programs and data are kept on a long time basis. It is cheaper than primary memory and slower than main or primary memory. It is non-volatile and cannot access data directly by the computer processor. It is the external memory of the computer system.

Example: hard disk drive, floppy disk, optical disk/ CD-ROM.

MEMORY CHIP ORGANISATION

A memory consists of cells in the form of an array. The basic element of the semiconductor memory is the **cell**. Each cell is capable of storing one bit of information. Each row of the cells constitutes a memory word and all cells of a row are connected to a common line referred to as a **word line**. An $A \times b$ memory has w words, each word having 'b' number of bits.

The basic memory element called cell can be in two states (0 or 1). The data can be written into the cell and can be read from it.

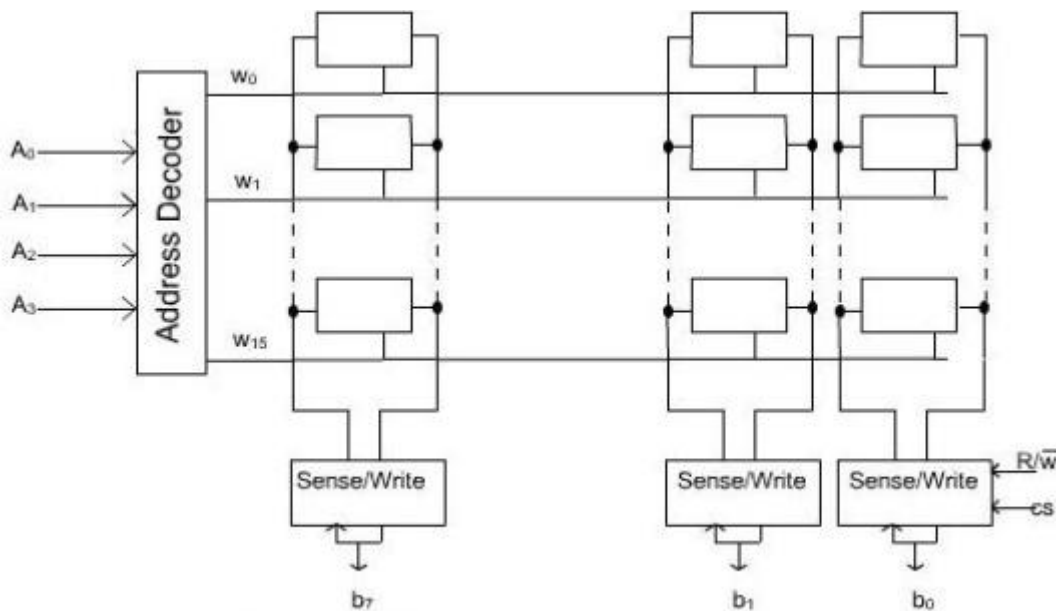


Fig 4.4: Organisation of 16 x 8 memory

- In the above diagram there are 16 memory locations named as $w_0, w_1, w_3, \dots, w_{15}$. Each location can store at most 8 bits of data ($b_0, b_1, b_3, \dots, b_7$). Each location (w_n) is the wordline. The wordline of Fig 4.4 is 8.
- Each row of the cell is a memory word. The memory words are connected to a common line termed as word line. The word line is activated based on the address it receives from the address bus.
- An address decoder is used to activate a word line.
- The cells in the memory are connected by two **bit lines** (column wise). These are connected to data input and data output lines through sense/ write circuitry.
- **Read Operation:** During read operation the sense/ write circuit reads the information by selecting the cell through wordline and bit lines. The data from this cell is transferred through the output data line.
- **Write Operation:** During write operation, the sense/ write circuitry gets the data and writes into the selected cell.
- The data input and output line of sense / write circuit is connected to a bidirectional data line.
- It is essential to have n bus lines to read 2^n words.

Organisation of 1M x 1 memory chip:

The organisation of 1024 x 1 memory chips, has 1024 memory words of size 1 bit only. The size of data bus is 1 bit and the size of address bus is 10 bits. A particular memory location is identified by the contents of memory address bus. A decoder is used to decode the memory address.

Organisation of memory word as a row:

- The whole memory address bus is used together to decode the address of the specified location.

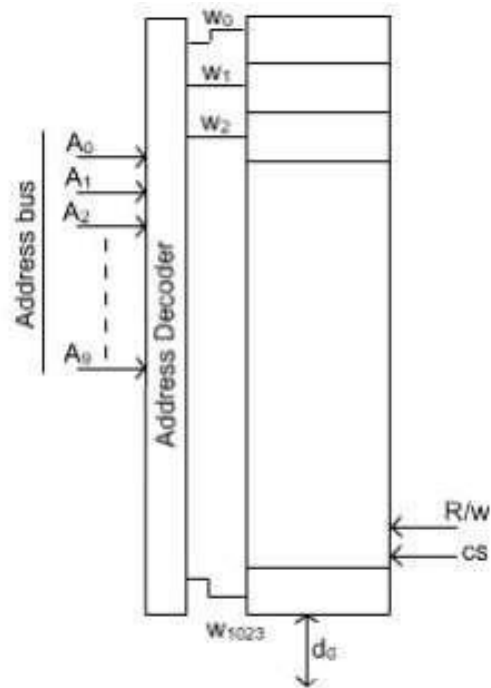


Fig 4.5: Organisation of memory word as row

Organisation of several memory words in row:

- One group is used to form the row address and the second group is used to form the column address.
- The 10-bit address is divided into two groups of 5 bits each to form the row and column address of the cell array.
- A row address selects a row of 32 cells, all of which could be accessed in parallel.
- Regarding the column address, only one of these cells is connected to the external data line via the input output multiplexers

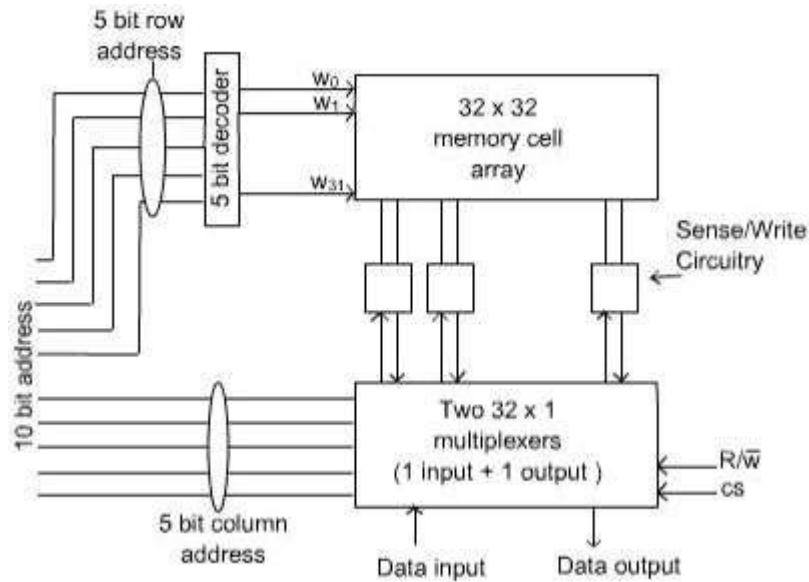


Fig 4.6: Organisation of several memory words in row

Signals used in memory chip:

- A memory unit of 1MB size is organised as 1M x 8 memory cells.
- It has got 2²⁰ memory location and each memory location contains 8 bits of information.
- The size of address bus is 20 and the size of data bus is 8.
- The number of pins of a memory chip depends on the data bus and address bus of the memory module.
- To reduce the number of pins required for the chip, the cells are organized in the form of a square array.
- The address bus is divided into two groups, one for column address and other one is for row address.
- In this case, high- and low-order 10 bits of 20-bit address constitute of row and column address of a given cell, respectively.
- In order to reduce the number of pin needed for external connections, the row and column addresses are multiplexed on ten pins.

- During a Read or a Write operation, the row address is applied first. In response to a signal pulse on the Row Address Strobe (RAS) input of the chip, this part of the address is loaded into the row address latch.
- All cells of this particular row are selected. Shortly after the row address is latched, the column address is applied to the address pins.
- It is loaded into the column address latch with the help of Column Address Strobe (CAS) signal, similar to RAS.
- The information in this latch is decoded and the appropriate Sense/Write circuit is selected.

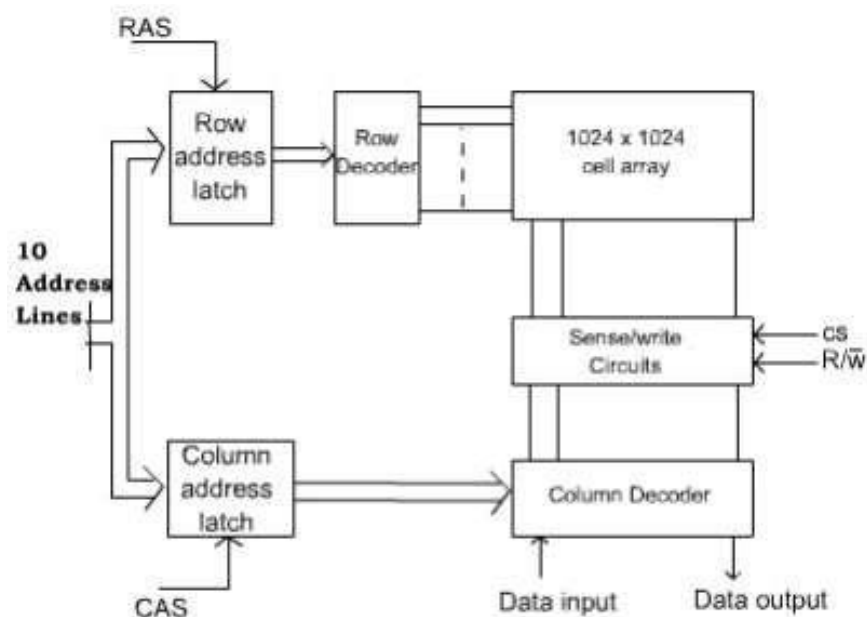


Fig 4.7: Signals in accessing the memory

- Each chip has a control input line called Chip Select (CS). A chip can be enabled to accept data input or to place the data on the output bus by setting its Chip Select input to 1.
- The address bus for the 64K memory is 16 bits wide.
- The high order two bits of the address are decoded to obtain the four chip select control signals.

- The remaining 14 address bits are connected to the address lines of all the chips.
- They are used to access a specific location inside each chip of the selected row.
- The R/ W inputs of all chips are tied together to provide a common read / write control.

CACHE MEMORY

The cache memory exploits the locality of reference to enhance the speed of the processor.

Cache memory or CPU memory, is high-speed SRAM that a processor can access more quickly than a regular RAM. This memory is integrated directly into the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU.

The cache memory stores instructions and data that are more frequently used or data that is likely to be used next. The processor looks first in the cache memory for the data. If it finds the instructions or data then it does perform a more time-consuming reading of data from larger main memory or other data storage devices.

The processor do not need to know the exact location of the cache. It can simply issue read and write instructions. The cache control circuitry determines whether the requested data resides in the cache.

- **Cache and temporal reference:** When data is requested by the processor, the data should be loaded in the cache and should be retained till it is needed again.
- **Cache and spatial reference:** Instead of fetching single data, a contiguous block of data is loaded into the cache.

Terminologies in Cache

- **Split cache:** It has separate data cache and a separate instruction cache. The two caches work in parallel, one transferring data and the other transferring instructions.
- **A dual or unified cache:**The data and the instructions are stored in the same cache. A combined cache with a total size equal to the sum of the two split caches will usually have a better hit rate.
- **Mapping Function:** The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

- **Cache Replacement:** When the cache is full and a memory word that is not in the cache is referenced, the cache control hardware must decide which block should be removed to create space for the new block that contains the referenced word. The collection of rules for making this decision is the replacement algorithm.

Cache performance:

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache. If the processor finds that the memory location is in the cache, a **cache hit** has said to be occurred. If the processor does not find the memory location in the cache, a **cache miss** has occurred. When a cache miss occurs, the cache replacement is made by allocating a new entry and copies in data from main memory. The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio**.

$$\text{Hit ratio} = \text{hit} / (\text{hit} + \text{miss}) = \text{Number of hits} / \text{Total accesses to the cache}$$

Miss penalty or cache penalty is the sum of time to place a block in the cache and time to deliver the block to CPU.

$$\text{Miss Penalty} = \text{time for block replacement} + \text{time to deliver the block to CPU}$$

Cache performance can be enhanced by using higher cache block size, higher associativity, reducing miss rate, reducing miss penalty, and reducing the time to hit in the cache. CPU execution Time of a given task is defined as the time spent by the system executing that task, including the time spent executing run-time or system services.

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{memory stall cycles (if any)}) \times \text{Clock cycle time}$$

The **memory stall cycles** are a measure of count of the memory cycles during which the CPU is waiting for memory accesses. This is dependent on caches misses and cost per miss (cache penalty).

$$\begin{aligned} \text{Memory stall cycles} &= \text{number of cache misses} \times \text{miss penalty} \\ &= \text{Instruction Count} \times (\text{misses/ instruction}) \times \text{miss penalty} \\ &= \text{Instruction Count (IC)} \times (\text{memory access/ instruction}) \times \text{miss penalty} \\ &= \text{IC} \times \text{Reads per instruction} \times \text{Read miss rate} \times \text{Read miss penalty} \\ &\quad + \text{IC} \times \text{Write per instruction} \times \text{Write miss rate} \times \text{Write miss penalty} \end{aligned}$$

Misses / instruction = (miss rate x memory access)/ instruction

Issues in Cache memory:

- **Cache placement:** where to place a block in the cache?
- **Cache identification:** how to identify that the requested information is available in the cache or not?
- **Cache replacement:** which block will be replaced in the cache, making way for an incoming block?

Cache Mapping Policies:

These policies determine the way of loading the main memory to the cache block. Main memory is divided into equal size partitions called as **blocks or frames**. The cache memory is divided into fixed size partitions called as **lines**. During cache mapping, block of main memory is copied to the cache and further access is made from the cache not from the main memory.

Cache mapping is a technique by which the contents of main memory are brought into the cache memory.

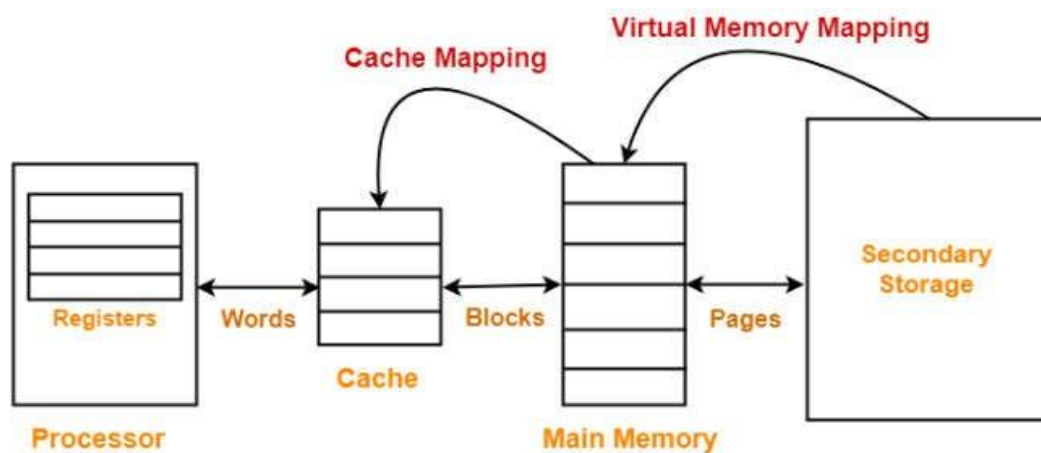


Fig 4.8: Cache mapping

There are three different cache mapping policies or mapping functions:

- i) Direct mapping
- ii) Fully Associative mapping
- iii) Set Associative mapping

Direct Mapping

- The simplest technique is direct mapping that maps each block of main memory into only one possible cache line.
- Here, each memory block is assigned to a specific line in the cache.
- If a line is previously taken up by a memory block and when a new block needs to be loaded, then the old block is replaced.
- Direct mapping's performance is directly proportional to the Hit ratio.

The direct mapping concept is if the i^{th} block of main memory has to be placed at the j^{th} block of cache memory $j = i \% (\text{number of blocks in cache memory})$

- Consider a 128 block cache memory. Whenever the main memory blocks 0, 128, 256 are loaded in the cache, they will be allotted cache block 0, since $j = (0 \text{ or } 128 \text{ or } 256) \% 128$ is zero).
- Contention or collision is resolved by replacing the older contents with latest contents.
- The placement of the block from main memory to the cache is determined from the 16 bit memory address.
- The lower order four bits are used to select one of the 16 words in the block.
- The 7 bit block field indicates the cache position where the block has to be stored.
- The 5 bit tag field represents which block of main memory resides inside the cache.
- This method is easy to implement but is not flexible.
- **Drawback:** The problem was that every block of main memory was directly mapped to the cache memory. This resulted in high rate of conflict miss. Cache memory has to be very frequently replaced even when other blocks in the cache memory were present as empty.

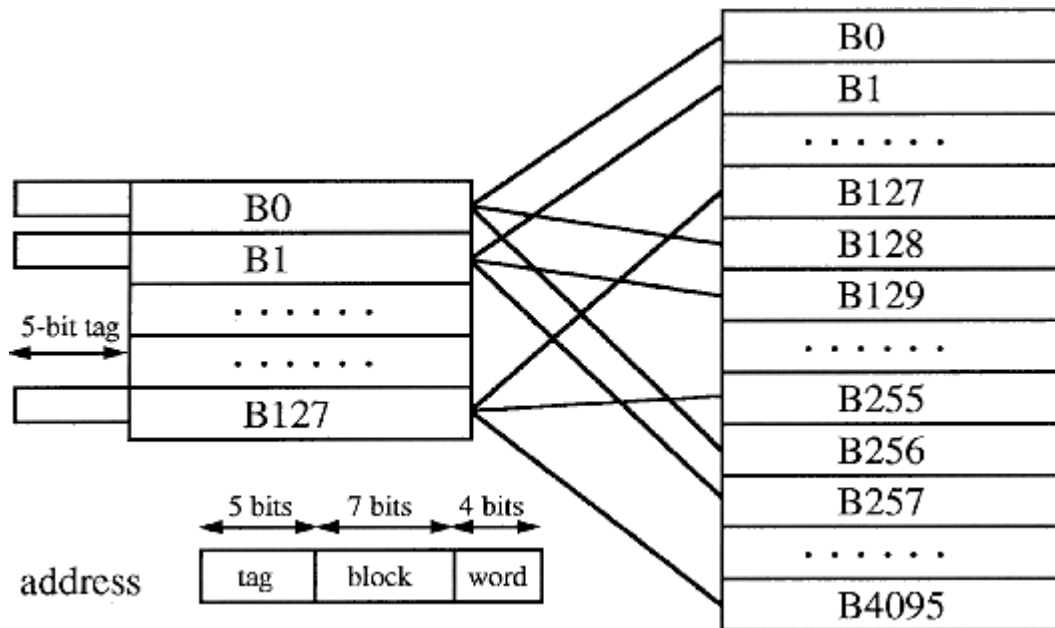


Fig 4.9: Direct memory mapping

Associative Mapping:

- The associative memory is used to store content and addresses of the memory word.
- Any block can go into any line of the cache. The 4 word id bits are used to identify which word in the block is needed and the remaining 12 bits represents the tag bit that identifies the main memory block inside the cache.
- This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.
- The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to check, if the desired block is present. Hence it is known as Associative Mapping technique.
- Cost of an associated mapped cache is higher than the cost of direct-mapped because of the need to search all 128 tag patterns to determine whether a block is in cache.

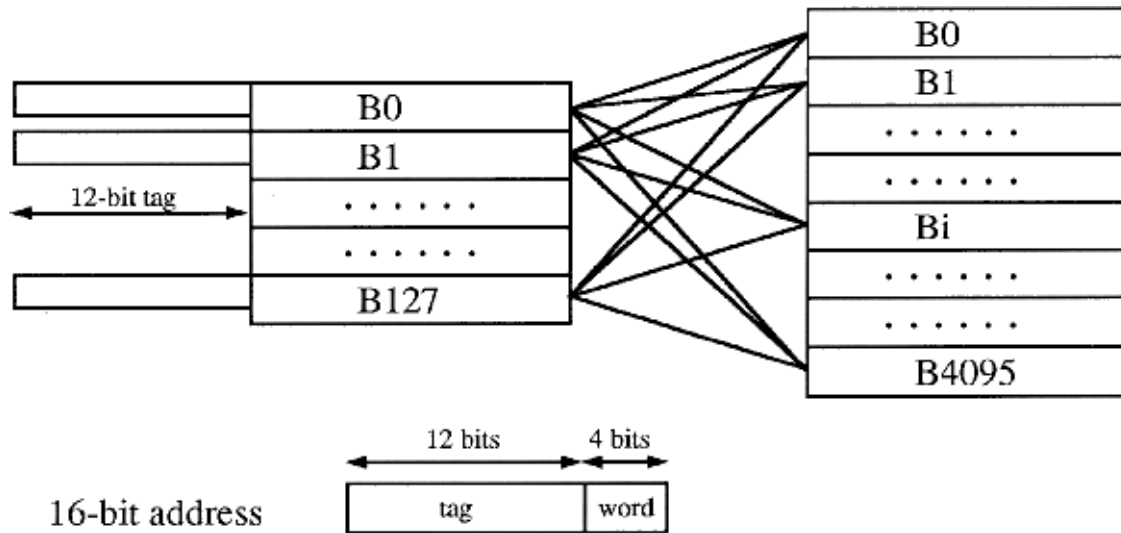


Fig 4.10: Associative Mapping

Set associative mapping:

- It is the combination of direct and associative mapping technique.
- Cache blocks are grouped into sets and mapping allow block of main memory to reside into any block of a specific set.
- This reduces contention problem (issue in direct mapping) with low hardware cost (issue in associative mapping).
- Consider a cache with two blocks per set. In this case, memory block 0, 64, 128,.....,4032 map into cache set 0 and they can occupy any two block within this set.
- It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a set. Then a block in memory can map to any one of the lines of a specific set.
- The 6 bit set field of the address determines which set of the cache might contain the desired block. The tag bits of address must be associatively compared to the tags of the two blocks of the set to check if desired block is present.

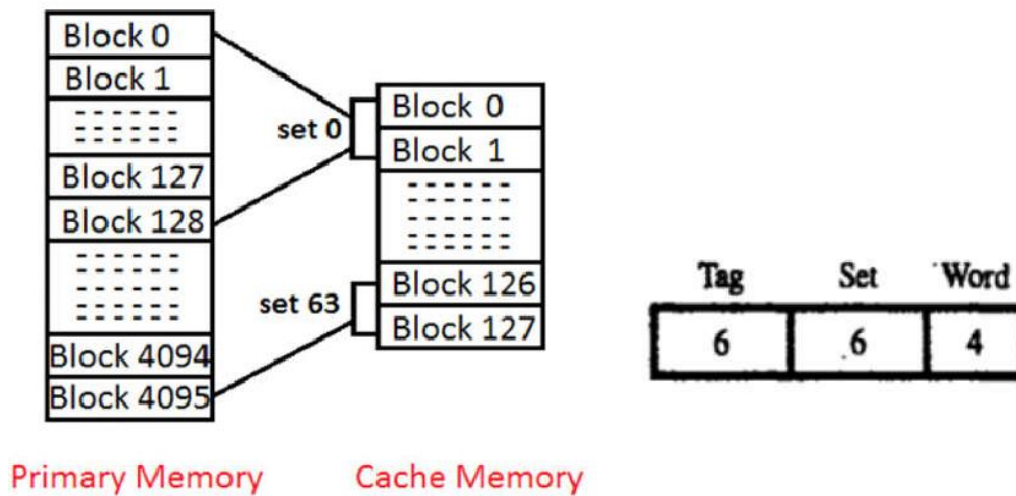


Fig 4.11: Set associative mapping

Handling Cache misses:

When a program accesses a memory location that is not in the cache, it is called a cache miss. The performance impact of a cache miss depends on the latency of fetching the data from the next cache level or main memory. The cache miss handling is done with the processor control unit and with a separate controller that initiates the memory access and refills the cache. The following are the steps taken when a cache miss occurs:

- Send the original PC value (PC - 4) to the memory.
- Instruct main memory to perform a read and wait for the memory to complete its access.
- Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.
- Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.

Writing to a cache:

- Suppose on a store instruction, the data is written into only the data cache (without changing main memory); then, after the write into the cache, memory would have a different value from that in the cache. This leads to inconsistency.

- The simplest way to keep the main memory and the cache consistent is to always write the data into both the memory and the cache. This scheme is called write-through.

Write through is a scheme in which writes always update both the cache and the memory, ensuring that data is always consistent between the two.

- With a write-through scheme, every write causes the data to be written to main memory. These writes will take a long time.
- A potential solution to this problem is deploying write buffer.
- A write buffer stores the data while it is waiting to be written to memory.
- After writing the data into the cache and into the write buffer, the processor can continue execution.
- When a write to main memory completes, the entry in the write buffer is freed.
- If the write buffer is full when the processor reaches a write, the processor must stall until there is an empty position in the write buffer.
- If the rate at which the memory can complete writes is less than the rate at which the processor is generating writes, no amount of buffering can help because writes are being generated faster than the memory system can accept them.

Write buffer is a queue that holds data while the data are waiting to be written to memory.

- The rate at which writes are generated may also be less than the rate at which the memory can accept them, and yet stalls may still occur. To reduce the occurrence of such stalls, processors usually increase the depth of the write buffer beyond a single entry.
- Another alternative to a write-through scheme is a scheme called write-back. When a write occurs, the new value is written only to the block in the cache.
- The modified block is written to the lower level of the hierarchy when it is replaced.
- Write-back schemes can improve performance, especially when processors can generate writes as fast or faster than the writes can be handled by main memory; a write-back scheme is, however, more complex to implement than write-through.

Write-back is a scheme that handles writes by updating values only to the block in the cache, then writing the modified block to the lower level of the hierarchy when the block is replaced.

Cache Replacement Algorithms

When a main memory block needs to be brought into the cache while all the blocks are occupied, then one of them has to be replaced. This selection of the block to be replaced is using cache replacement algorithms. Replacement algorithms are only needed for associative and set associative techniques. The following are the common replacement techniques:

- **Least Recently Used (LRU):** This replaces the cache line that has been in the cache the longest with no references to it.
- **First-in First-out (FIFO):** This replaces the cache line that has been in the cache the longest.
- **Least Frequently Used (LFU):** This replaces the cache line that has experienced the fewest references.
- **Random:** This picks a line at random from the candidate lines.

Example 4.1: Program P runs on computer A in 10 seconds. Designer says clock rate can be increased significantly, but total cycle count will also increase by 20%. What clock rate do we need on computer B for P to run in 6 seconds? (Clock rate on A is 100 MHz).

The new machine is B. We want CPU Time_B = 6 seconds.

We know that Cycles count_B = 1.2 Cycles count_A. Calculate Cycles count_A.

CPU Time_A = 10 sec. = ; Cycles count_A = 1000 x 10⁶ cycles

Calculate Clock rate_B:

CPU Time_B = 6 sec. = ; Clock rate_B = = 200 MHz

Machine B must run at twice the clock rate of A to achieve the target execution time.

Example 4.2: We have two machines with different implementations of the same ISA. Machine A has a clock cycle time of 10 ns and a CPI of 2.0 for program P; machine B has a clock cycle time of 20 ns and a CPI of 1.2 for the same program. Which machine is faster?

Let IC be the number of instructions to be executed. Then

Cycles count_A = 2.0 IC

Cycles count_B = 1.2 IC

calculate CPU Time for each machine:

CPU Time_A = 2.0 IC x 10 ns = 20.0 IC ns

CPU Time_B = 1.2 IC x 20 ns = 24.0 IC ns

» Machine A is 20% faster.

Example 4.3: Consider an implementation of MIPS ISA with 500 MHz clock and

- each ALU instruction takes 3 clock cycles,
- each branch/jump instruction takes 2 clock cycles,
- each sw instruction takes 4 clock cycles,
- eachlw instruction takes 5 clock cycles.

Also, consider a program that during its execution executes:

- x=200 million ALU instructions
- y=55 million branch/jump instructions
- z=25 million sw instructions
- w=20 million lw instructions

Find CPU time. Assume sequentially executing CPU.

$$\begin{aligned} \text{Clock cycles for a program} &= (3x + 2y + 4z + 5w) \\ &= 910 \times 10^6 \text{ clock cycles} \end{aligned}$$

$$\begin{aligned} \text{CPU_time} &= \text{Clock cycles for a program} / \text{Clock rate} \\ &= 910 \times 10^6 / 500 \times 10^6 = 1.82 \text{ sec} \end{aligned}$$

Example 4.4: Consider another implementation of MIPS ISA with 1 GHz clock and

- each ALU instruction takes 4 clock cycles,
- each branch/jump instruction takes 3 clock cycles,
- each sw instruction takes 5 clock cycles,
- eachlw instruction takes 6 clock cycles.

Also, consider the same program as in Example 1.

Find CPI and CPU time. Assume sequentially executing CPU.

$$\begin{aligned} \text{CPI} &= (4x + 3y + 5z + 6w) / (x + y + z + w) \\ &= 4.03 \text{ clock cycles/ instruction} \end{aligned}$$

$$\begin{aligned} \text{CPU time} &= \text{Instruction count} \times \text{CPI} / \text{Clock rate} \\ &= (x+y+z+w) \times 4.03 / 1000 \times 10^6 \\ &= 300 \times 10^6 \times 4.03 / 1000 \times 10^6 \\ &= 1.21 \text{ sec} \end{aligned}$$

VIRTUAL MEMORY

Virtual memory is a memory management capability of an operating system that uses hardware and software to allow a computer to compensate for physical memory shortages by temporarily transferring data from RAM to disk storage.

The concept of virtual memory in computer organisation is allocating memory from the hard disk and making that part of the hard disk as a temporary RAM. In other words, it is a technique that uses main memory as a cache for secondary storage. The motivations for virtual memory are:

- To allow efficient and safe sharing of memory among multiple programs
- To remove the programming burdens of a small, limited amount of main memory.

Virtual memory provides an illusion to the users that the PC has enough primary memory left to run the programs. Sometimes the size of programs to be executed may sometimes very bigger than the size of primary memory left, the user never feels that the system needs a bigger primary storage to run that program. When the RAM is full, the operating system occupies a portion of the hard disk and uses it as a RAM. In that part of the secondary storage, the part of the program which not currently being executed is stored and all the parts of the program that are executed are first brought into the main memory. This is the theory behind **virtual memory**.

Terminologies:

- **Physical address** is an address in main memory.
- **Protection** is a set of mechanisms for ensuring that multiple processes sharing the processor, memory, or I/O devices cannot interfere, with one another by reading or writing each other's data.
- Virtual memory breaks programs into fixed-size blocks called **pages**.
- **Page fault** is an event that occurs when an accessed page is not present in main memory.
- **Virtual address** is an address that corresponds to a location in virtual space and is translated by address mapping to a physical address when memory is accessed.
- **Address translation or address mapping** is the process by which a virtual address is mapped to an address used to access memory.

Working mechanism

- In virtual memory, blocks of memory are mapped from one set of addresses (virtual addresses) to another set (physical addresses).

- The processor generates virtual addresses while the memory is accessed using physical addresses.
- Both the virtual memory and the physical memory are broken into pages, so that a virtual page is really mapped to a physical page.
- It is also possible for a virtual page to be absent from main memory and not be mapped to a physical address, residing instead on disk.
- Physical pages can be shared by having two virtual addresses point to the same physical address. This capability is used to allow two different programs to share data or code.
- Virtual memory also simplifies loading the program for execution by providing relocation. **Relocation** maps the virtual addresses used by a program to different physical addresses before the addresses are used to access memory. This relocation allows us to load the program anywhere in main memory.

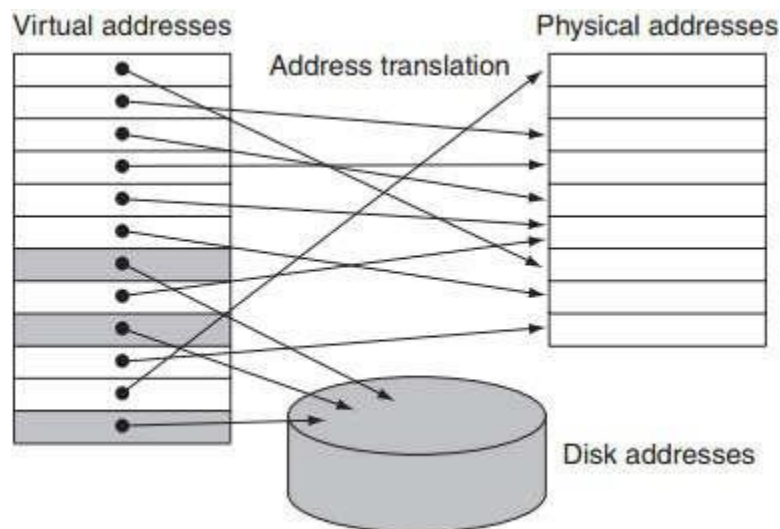


Fig 4.12: Mapping of virtual and physical memory

Addressing in virtual memory

- A virtual address is considered as a pair (p,d) where lower order bits give an offset d within the page and high-order bits specify the page p .
- The job of the Memory Management Unit (MMU) is to translate the page number p to a frame number f .

- The physical address is then (f,d), and this is what goes on the memory bus.
- For every process, there is a page and page-number p is used as an index into this array for the translation.
- The following are the entries in page tables:
 1. Validity bit: Set to 0 if the corresponding page is not in memory
 2. Frame number: Number of bits required depends on size of physical memory
 3. Protection bits: Read, write, execute accesses
 4. Referenced bit is set to 1 by hardware when the page is accessed: used by page replacement policy
 5. Modified bit (dirty bit) set to 1 by hardware on write-access: used to avoid writing when swapped out.

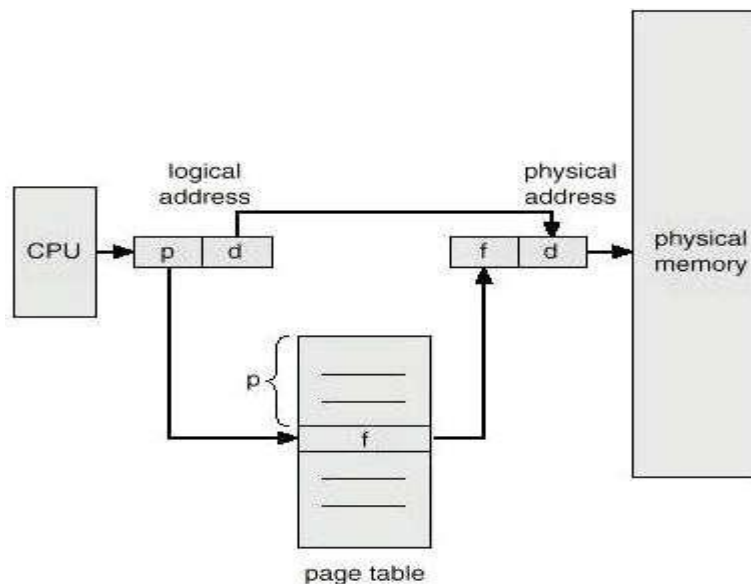


Fig 4.13: Conversion of logical address to physical address

Role of control bit in page table

The control bit (v) indicates whether the page is loaded in the main memory. It also indicates whether the page has been modified during its residency in the main memory. This information is crucial to determine whether to write back the page to the disk before it is removed from the main memory during next page replacement.

	frame number	valid-invalid bit
0	2	v
1	3	v
2	4	v
3	7	v
4	8	v
5	9	v
6	0	i
7	0	i

page table

Fig 4.14: Page table

Page faults and page replacement algorithms

A page fault occurs when a page referenced by the CPU is not found in the main memory. The required page has to be brought from the secondary memory into the main memory. A page that is currently residing in the main memory, has to be replaced if all the frames of main memory are already occupied.

Page replacement is a process of swapping out an existing page from the frame of a main memory and replacing it with the required page.

Page replacement is done when all the frames of main memory are already occupied and a page has to be replaced to create a space for the newly referenced page. A good replacement algorithm will have least number of page faults.

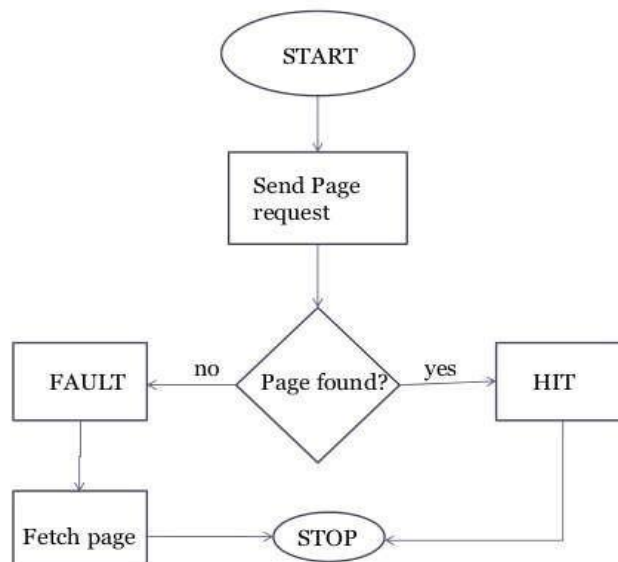


Fig 4.14: Occurrence of page fault

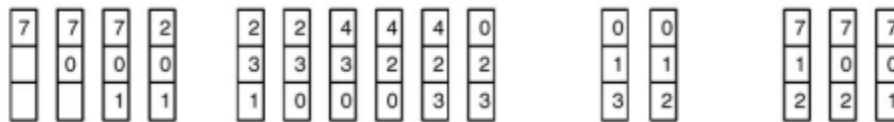
The following are the page replacement algorithms:

1. FIFO Page Replacement Algorithm
2. LIFO Page Replacement Algorithm
3. LRU Page Replacement Algorithm
4. Optimal Page Replacement Algorithm
5. Random Page Replacement Algorithm

1. First In First Out (FIFO) page replacement algorithm

It replaces the oldest page that has been present in the main memory for the longest time. It is implemented by keeping track of all the pages in a queue.

Example 4.5. Find the page faults when the following pages are requested to be loaded in a page frame of size 3: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



Page faults= 15

2. Last In First Out (LIFO) page replacement algorithm

It replaces the newest page that arrived at last in the main memory. It is implemented by keeping track of all the pages in a stack.

3. Least Recently Used (LRU) page replacement algorithm

The new page will be replaced with least recently used page.

Example 4.6: Consider the following reference string. Calculate the number of page faults when the page frame size is 3 using LRU policy. 7, 0, 1, 2, 0, 3, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
F	F	F	F		F		F	F	F	F			F		F		F		

Page faults= 12 (F bit indicates the occurrence of page faults)

4. Optimal page replacement algorithm

In this method, pages are replaced which would not be used for the longest duration of time in the future.

Example 4.7: Find the number of misses and hits while using optimal page replacement algorithm on the following reference string with page frame size as 4: 2, 3, 4, 2, 1, 3, 7, 5, 4, 3, 2, 3, 1.

2	2	2	2	2	2	2	2	2	2	2	2	1
	3	3	3	3	3	3	3	3	3	3	3	3
		4	4	4	4	4	4	4	4	4	4	4
			1	1	7	5	5	5	5	5	5	5

Page fault=13 Number of page hit= 6 Number of page misses=7

5. Random page replacement algorithms

Random replacement algorithm replaces a random page in memory. This eliminates the overhead cost of tracking page references.

Translation Lookaside Buffer (TLB)

A translation lookaside buffer (TLB) is a memory cache that stores recent translations of virtual memory to physical addresses for faster retrieval.

The page tables are stored in main memory and every memory access by a program to the page table takes longer time. This is because it does one memory access to obtain the physical address and a second access to get the data. The virtual to physical memory address translation occurs twice. But a TLB will exploit the locality of reference and can reduce the memory access time.

TLB hit is a condition where the desired entry is found in translation look aside buffer. If this happens then the CPU simply access the actual location in the main memory.

If the entry is not found in TLB (TLB miss) then CPU has to access page table in the main memory and then access the actual frame in the main memory. Therefore, in the case of TLB hit, the effective access time will be lesser as compare to the case of TLB miss.

If the probability of TLB hit is P% (TLB hit rate) then the probability of TLB miss (TLB miss rate) will be (1-P) %. The effective access time can be defined as

$$\text{Effective access time} = P(t + m) + (1 - p)(t + k.m + m)$$

Where, p is the TLB hit rate, t is the time taken to access TLB, m is the time taken to access main memory. K indicates the single level paging has been implemented.

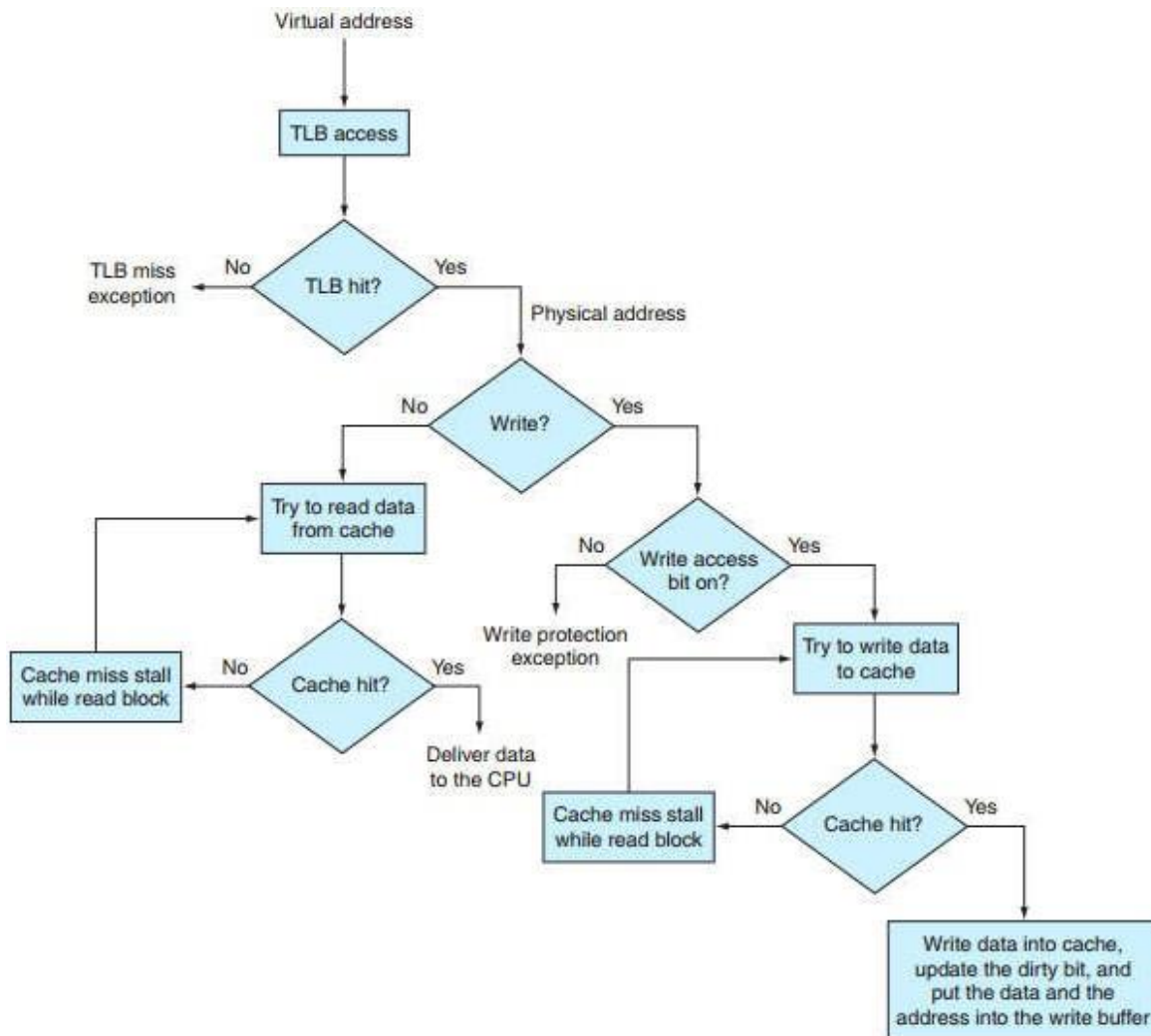


Fig 4.15: Cache access levels

4.3.5 Protection in Virtual memory

- Virtual memory allows sharing of main memory by multiple processes. So protection mechanisms, while providing memory protection.
- The protection mechanism must ensure one process cannot write into the address space of another user process or into the operating system.
- Memory protection can be done at two levels: hardware and software levels.

Hardware Level:

Memory protection at hardware level is done in three methods:

- The machine should support two modes: supervisor mode and user mode. This indicates whether the current running process is a user or supervisory process. The processes running in supervisor or kernel mode is an operating system process.
- Include user / supervisor bit in TLB to indicate whether the process is in user or supervisor mode. This is an access control mechanism imposed on the user process only to read from the TLB and not write to it.
- The processors can switch between user and supervisor mode. The switching from user to system mode is done through system calls, that transfers control to a dedicated location in supervisor code space.

System call is a special instruction that transfers control from user mode to a dedicated location in supervisor code space, invoking the exception mechanism in the process.

PARALLEL BUS ARCHITECTURES

Single bus architectures connect multiple processors with their own cache memory using shared bus. This is a simple architecture but it suffers from latency and bandwidth issues. This naturally led to deploying parallel or multiple bus architectures. Multiple bus multiprocessor systems use several parallel buses to interconnect multiple processors with multiple memory modules. The following are the connection schemes in multi bus architectures:

1. Multiple-bus with full bus–memory connection (MBFBMC)

This has all memory modules connected to all buses. The multiple-bus with single bus–

memory connection has each memory module connected to a specific bus. For N processors with M memory modules and B buses, the number of connections requires are: $B(N+M)$ and the load on each bus will be $N+M$.

2. Multiple bus with partial bus-memory connection (MBPBMC)

The multiple-bus with partial bus-memory connection, has each memory module connected to a subset of buses.

3. Multiple bus with class-based memory connection (MBCBMC)

The multiple-bus with class-based memory connection (MBCBMC), has memory modules grouped into classes whereby each class is connected to a specific subset of buses. A class is just an arbitrary collection of memory modules.

4. Multiple bus with single bus memory connection (MBSBMC)

Here, only single bus will be connected to single memory, but the processor can access all the buses. The numbers of connections:

$$BN + \sum_{j=1}^k M_j(j + B - k)$$

and load on each bus is given by

$$N + \sum_{j=\max(i+k-B,1)}^k M_j, 1 \leq i \leq B$$

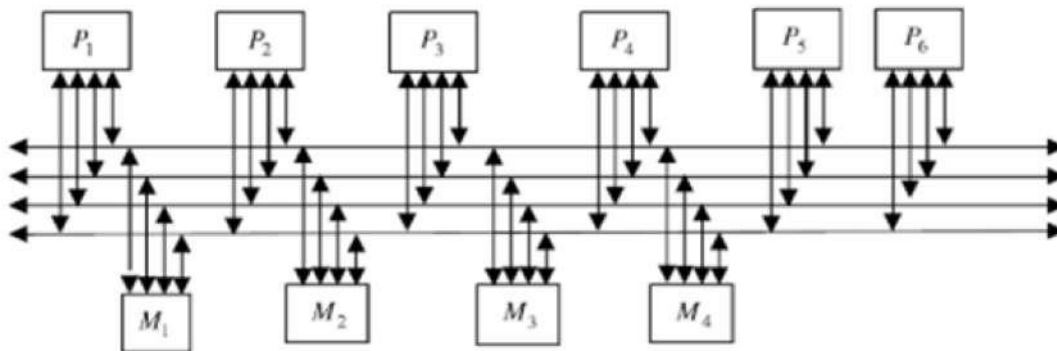


Fig 4.16 a) Multiple-bus with full bus-memory connection (MBFBMC)

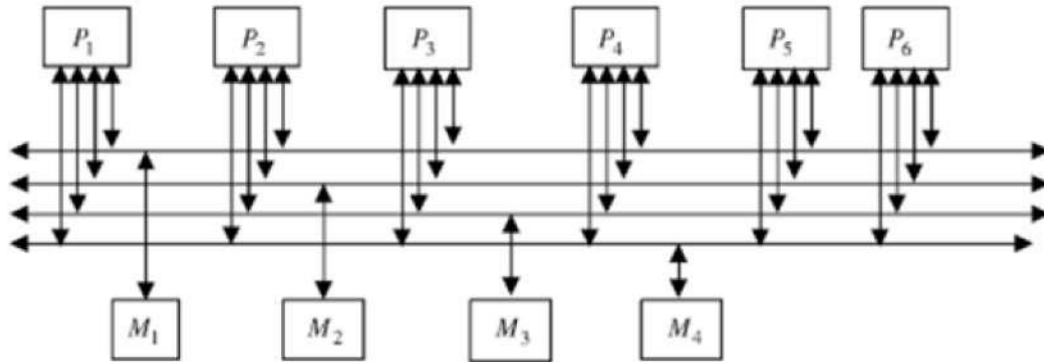


Fig 4.16 b) Multiple bus with single bus memory connection (MBSBMC)

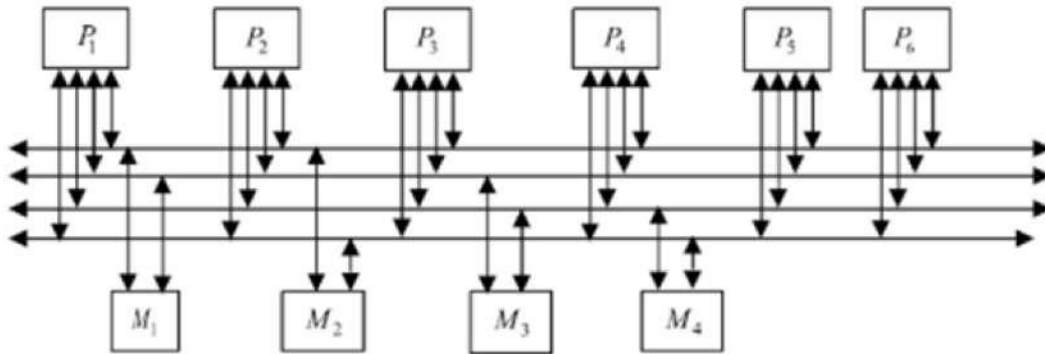


Fig 4.16 c) Multiple bus with partial bus-memory connection (MBPBMC)

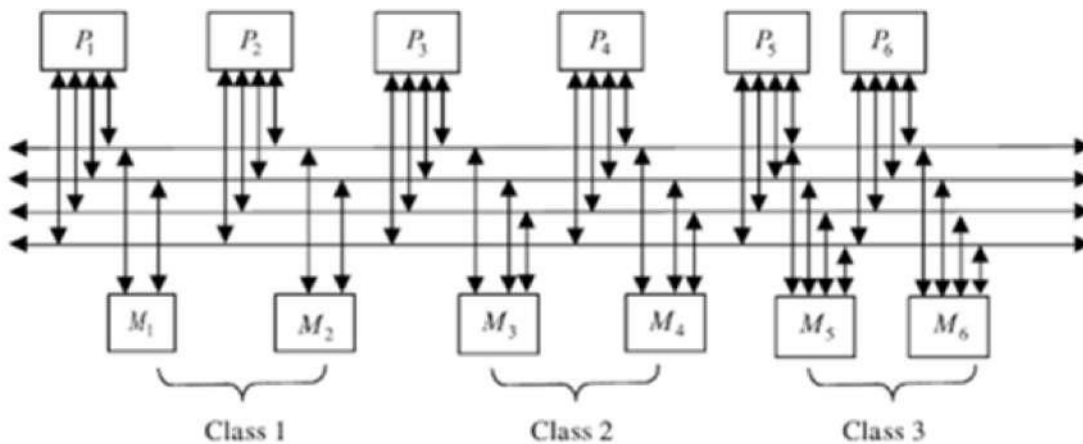


Fig 4.16 d) Multiple bus with class-based memory connection (MBCBMC)

Bus Synchronisation:

- In a single bus multiprocessor system, bus arbitration is required in order to resolve the bus contention that takes place when more than one processor competes to access the bus.
- A bus can be classified as synchronous or asynchronous. The time for any transaction over a synchronous bus is known in advance. Asynchronous bus depends on the availability of data and the readiness of devices to initiate bus transactions.
- The processors that want to use the bus submit their requests to bus arbitration logic. The latter decides, using a certain priority scheme, which processor will be granted access to the bus during a certain time interval (bus master).
- The process of passing bus mastership from one processor to another is called **handshaking**, which requires the use of two control signals: busrequest and bus grant.
- Bus request indicates that a given processor is requesting mastership of the bus.
- Bus grant: indicates that bus mastership is granted.
- Bus busy: is usually used to indicate whether or not the bus is currently being used.
- In deciding which processor gains control of the bus, the bus arbitration logic uses a predefined priority scheme.
- Among the priority schemes used are random priority, simple rotating priority, equal priority, and least recently used (LRU) priority.
- After each arbitration cycle, in simple rotating priority, all priority levels are reduced one place, with the lowest priority processor taking the highest priority. In equal priority, when two or more requests are made, there is equal chance of any one request being processed.
- In the LRU algorithm, the highest priority is given to the processor that has not used the bus for the longest time.

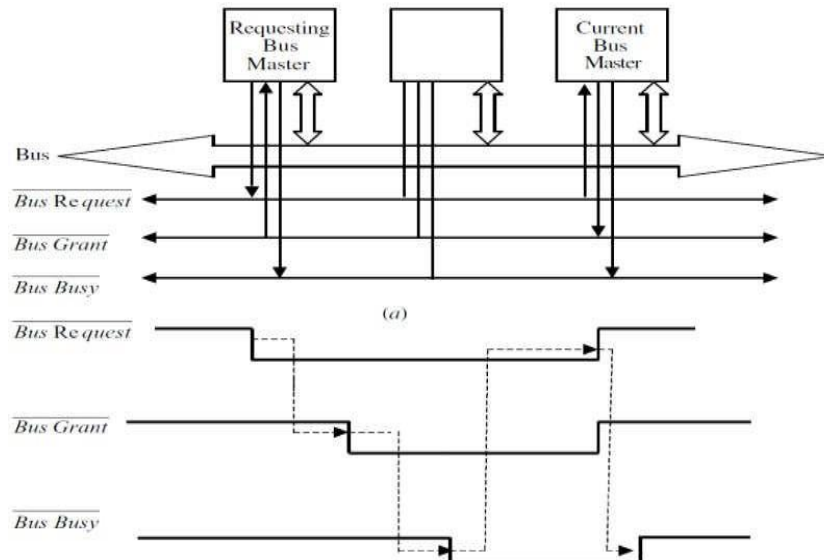


Fig 4.17: Bus synchronisation

INTERNAL COMMUNICATION METHODOLOGIES

CPU of the computer system communicates with the memory and the I/O devices in order to transfer data between them. The method of communication of the CPU with memory and I/O devices is different. The CPU may communicate with the memory either directly or through the Cache memory. However, the communication between the CPU and I/O devices is usually implemented with the help of interface. There are three types of internal communications:

- Programmed I/O
- Interrupt driven I/O
- Direct Memory Access (DMA)

Programmed I/O

- Programmed I/O is implicated to data transfers that are initiated by a CPU, under driver software control to access Registers or Memory on a device.
- With programmed I/O, data are exchanged between the processor and the I/O module.

- The processor executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command, and transferring the data.
- When the processor issues a command to the I/O module, it must wait until the I/O operation is complete.
- If the processor is faster than the I/O module, this is wasteful of processor time. With interrupt-driven I/O, the processor issues I/O command, continues to execute other instructions, and is interrupted by the I/O module when the latter has completed its work.
- With both programmed and interrupt I/O, the processor is responsible for extracting data from main memory for output and storing data in main memory for input.
- The alternative is known as direct memory access. In this mode, the I/O module and main memory exchange data directly, without processor involvement.
- With programmed I/O, the I/O module will perform the requested action and then set the appropriate bits in the I/O status register.
- The I/O module takes no further action to alert the processor.
- When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module. In particular, it does not interrupt the processor.
- It is the responsibility of the processor periodically to check the status of the I/O module. Then if the device is ready for the transfer (read/write).
- The processor transfers the data to or from the I/O device as required. As the CPU is faster than the I/O module, the problem with programmed I/O is that the CPU has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data.
- The CPU, while waiting, must repeatedly check the status of the I/O module, and this process is known as **Polling**.
- The level of the performance of the entire system is severely degraded.

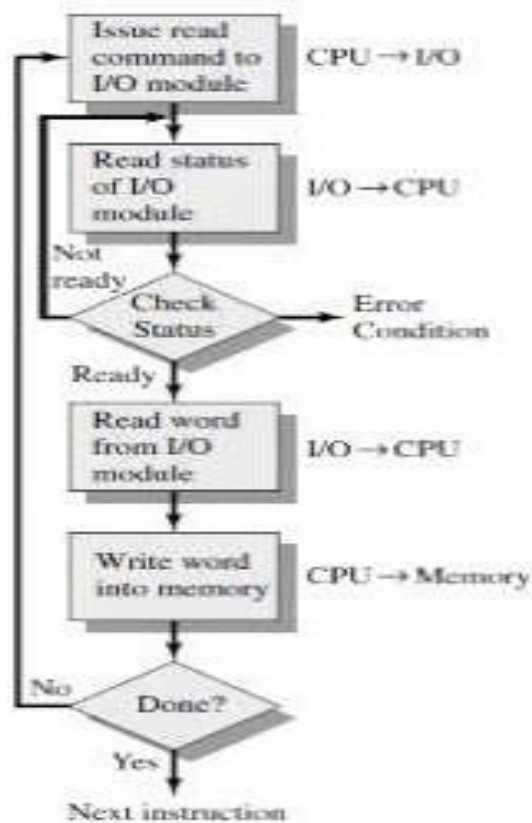


Fig 4.18: Workflow in programmed I/O

Interrupt Driven I/O

- The CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.
- For input, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports, memory mapping.
- For output, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.
- Although Interrupt relieves the CPU of having to wait for the devices, but it is still inefficient in data transfer of large amount because the CPU has to transfer the data word by word between I/O module and memory.
- Below are the basic operations of Interrupt:

1. CPU issues read command
2. I/O module gets data from peripheral whilst CPU does other work
3. I/O module interrupts CPU
4. CPU requests data
5. I/O module transfers data

Direct Memory Access (DMA)

- Direct Memory Access (DMA) means CPU grants I/O module authority to read from or write to memory without involvement.
- DMA module controls exchange of data between main memory and the I/O device.
- Because of DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus relieve congestion on the bus.
- CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred.

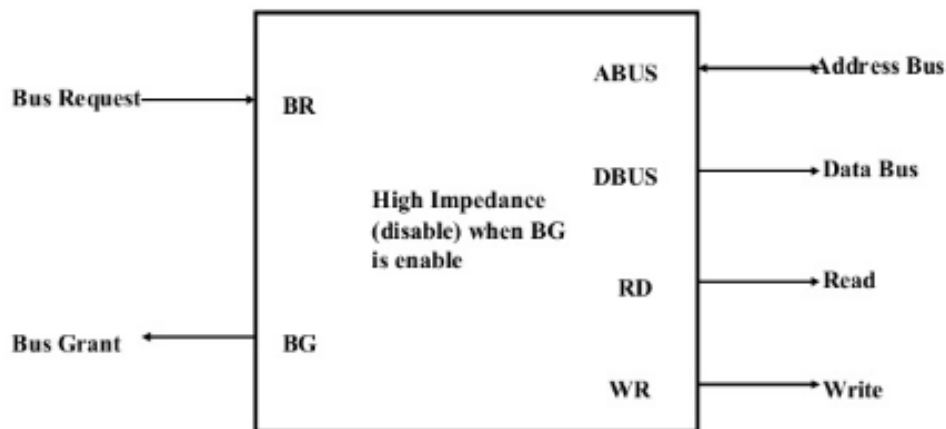


Fig 4.19: CPU bus signals for DMA transfer

- The CPU programs the DMA controller by setting its registers so it knows what to transfer where.
- It also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum.

- When valid data are in the disk controller's buffer, DMA can begin. The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller.
- This read request looks like any other read request, and the disk controller does not know whether it came from the CPU or from a DMA controller.
- The memory address to write to is on the bus address lines, so when the disk controller fetches the next word from its internal buffer, it knows where to write it.
- The write to memory is another standard bus cycle.
- When the write is complete, the disk controller sends an acknowledgement signal to the DMA controller, also over the bus.
- The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0, steps 2 through 4 are repeated until the count reaches 0.
- At that time, the DMA controller interrupts the CPU to let it know that the transfer is now complete.
- When the operating system starts up, it does not have to copy the disk block to memory; it is already there.
- The DMA controller requests the disk controller to transfer data from the disk controller's buffer to the main memory. In the first step, the CPU issues a command to the disk controller telling it to read data from the disk into its internal buffer.

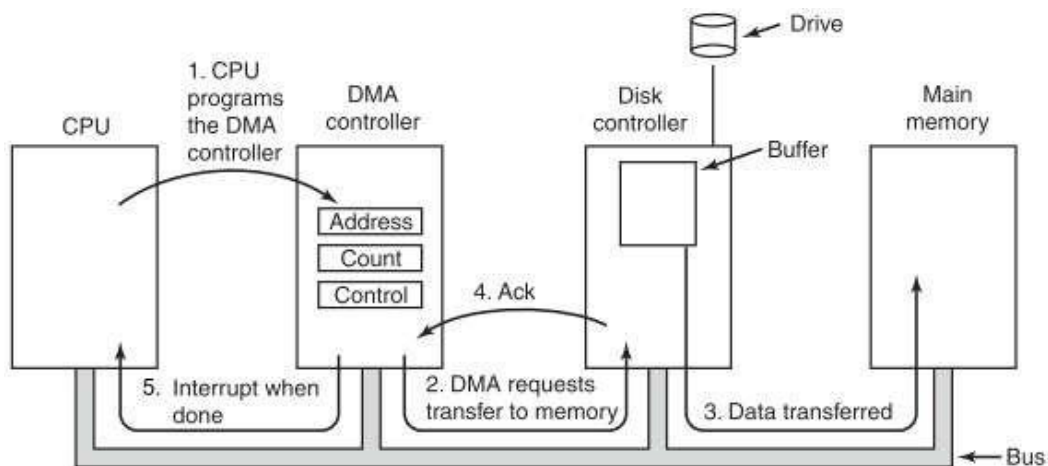


Fig 4.20: Operations in DMA

SERIAL BUS ARCHITECTURES

The peripheral devices and external buffer that operate at relatively low frequencies communicate with the processor using serial bus. There are two popular serial buses: Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I²C).

Serial Peripheral Interface (SPI)

Serial Peripheral Interface (SPI) is an interface bus designed by Motorola to send data between microcontrollers and small peripherals such as shift registers, sensors, and SD cards. It uses separate clock and data lines, along with a select line to choose the device.

- A standard SPI connection involves a master connected to slaves using the serial clock (SCK), Master Out Slave In (MOSI), Master In Slave Out (MISO), and Slave Select (SS) lines.
- The SCK, MOSI, and MISO signals can be shared by slaves while each slave has a unique SS line.
- The SPI interface defines no protocol for data exchange, limiting overhead and allowing for high speed data streaming.
- Clock polarity (CPOL) and clock phase (CPHA) can be specified as '0' or '1' to form four unique modes to provide flexibility in communication between master and slave.
- If CPOL and CPHA are both '0' (defined as Mode 0) data is sampled at the leading rising edge of the clock. Mode 0 is by far the most common mode for SPI bus slave communication.
- If CPOL is '1' and CPHA is '0' (Mode 2), data is sampled at the leading falling edge of the clock.
- Likewise, CPOL = '0' and CPHA = '1' (Mode 1) results in data sampled at on the trailing falling edge and CPOL = '1' with CPHA = '1' (Mode 3) results in data sampled on the trailing rising edge.

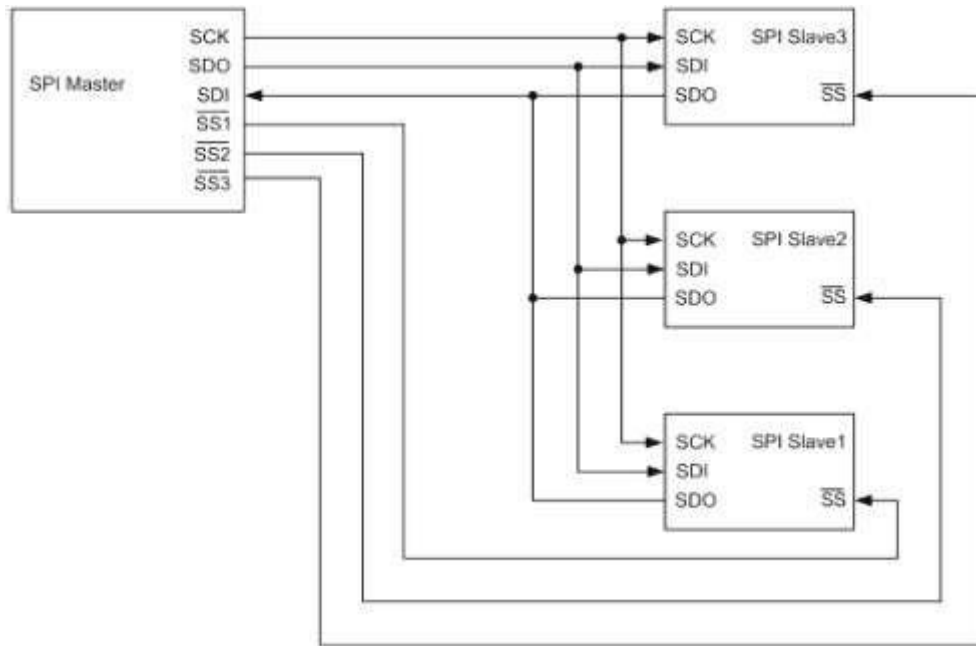


Fig 4.21: SPI master with three slaves

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

Fig 4.22: Modes in SPI

- In addition to the standard 4-wire configuration, the SPI interface has been extended to include a variety of IO standards including 3-wire for reduced pin count and dual or quad I/O for higher throughput.
- In 3-wire mode, MOSI and MISO lines are combined to a single bidirectional data line.
- Transactions are half-duplex to allow for bidirectional communication. Reducing the number of data lines and operating in half-duplex mode also decreases maximum possible throughput; many 3-wire devices have low performance requirements and are instead designed with low pin count in mind.

- Multi I/O variants such as dual I/O and quad I/O add additional data lines to the standard for increased throughput.
- Components that utilize multi I/O modes can rival the read speed of parallel devices while still offering reduced pin counts. This performance increase enables random access and direct program execution from flash memory (execute-in-place).

Inter-Integrated Circuit (I²C)

An inter-integrated circuit (Inter-IC or I2C) is a multi-master serial bus that connects low-speed peripherals to a motherboard, mobile phone, embedded system or other electronic devices.

- Philips Semiconductor created I²C with an intention of communication between chips reside on the same Printed Circuit Board (PCB).
- It is a multi-master, multi-slave protocol.
- It is designed to lessen costs by streamlining massive wiring systems with an easier interface for connecting a central processing unit (CPU) to peripheral chips in a television.
- It had a battery-controlled interface but later utilized an internal bus system.
- It is built on two lines
- ✦ SDA (Serial Data) – The line for the master and slave to send and receive data
- ✦ SCL (Serial Clock) – The line that carries the clock signal.
- Devices on an I2C bus are always a master or a slave. Master is the device which always initiates a communication and drives the clock line (SCL). Usually a microcontroller or microprocessor acts a master which needs to read data from or write data to slave peripherals.
- Slave devices are always responds to master and won't initiate any communication by itself. Devices like EEPROM, LCDs, RTCs acts as a slave device. Each slave device will have a unique address such that master can request data from or write data to it.
- The master device uses either a 7-bit or 10-bit address to specify the slave device as its partner of data communication and it supports bi-directional data transfer.

Working of I²C

- The I²C, data is transferred in messages, which are broken up into frames of data. Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted.
- The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame.
- The following are the bits in data frames:
 1. **Start Condition:** The SDA line switches from a high voltage level to a low voltage level before the SCL line switches from high to low.
 2. **Stop Condition:** The SDA line switches from a low voltage level to a high voltage level after the SCL line switches from low to high.
 3. **Address Frame:** A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.
 4. **Read/Write Bit:** A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).
 5. **ACK/NACK Bit:** Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.

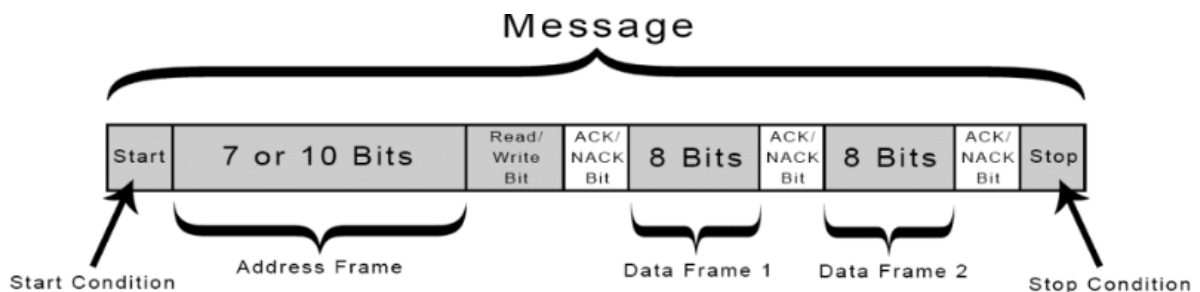


Fig 4.23: I²C Message Format

Addressing:

- I²C doesn't have slave select lines like SPI, so it needs another way to let the slave know that data is being sent to it, and not another slave. It does this by addressing. The address frame is always the first frame after the start bit in a new message.

- The master sends the address of the slave it wants to communicate with to every slave connected to it. Each slave then compares the address sent from the master to its own address.
- If the address matches, it sends a low voltage ACK bit back to the master. If the address doesn't match, the slave does nothing and the SDA line remains high.

Read/Write Bit

- The address frame includes a single bit at the end that informs the slave whether the master wants to write data to it or receive data from it. If the master wants to send data to the slave, the read/write bit is a low voltage level. If the master is requesting data from the slave, the bit is a high voltage level.

Data Frame

- After the master detects the ACK bit from the slave, the first data frame is ready to be sent.
- The data frame is always 8 bits long, and sent with the most significant bit first.
- Each data frame is immediately followed by an ACK/NACK bit to verify that the frame has been received successfully.
- The ACK bit must be received by either the master or the slave (depending on who is sending the data) before the next data frame can be sent.
- After all of the data frames have been sent, the master can send a stop condition to the slave to halt the transmission.
- The stop condition is a voltage transition from low to high on the SDA line after a low to high transition on the SCL line, with the SCL line remaining high.

Steps in Data transmission

1. The master sends the start condition to every connected slave by switching the SDA line from a high voltage level to a low voltage level before switching the SCL line from high to low.
2. The master sends each slave the 7 or 10 bit address of the slave it wants to communicate with, along with the read/write bit.

3. Each slave compares the address sent from the master to its own address. If the address matches, the slave returns an ACK bit by pulling the SDA line low for one bit. If the address from the master does not match the slave's own address, the slave leaves the SDA line high.
4. The master sends or receives the data frame.
5. After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful receipt of the frame.
6. To stop the data transmission, the master sends a stop condition to the slave by switching SCL high before switching SDA high.

Advantages

- It uses two wires.
- This supports multiple masters and multiple slaves.
- ACK/NACK bit gives confirmation that each frame is transferred successfully.
- Well known and widely used protocol

Disadvantages

- Slower data transfer rate than SPI.
- The size of the data frame is limited to 8 bits
- More complicated hardware needed to implement than SPI.

MASS STORAGE

Mass storage refers to various techniques and devices for storing large amounts of data. Mass storage is distinct from memory, which refers to temporary storage areas within the computer. Unlike main memory, mass storage devices retain data even when the computer is turned off.

The mass storage medium includes:

- solid-state drives (SSD)
- hard drives
- external hard drives
- optical drives

- tape drives
- RAID storage
- USB storage
- flash memory cards

Solid State Devices

- Solid-state devices are electronic devices in which electricity flows through solid semiconductor crystals like silicon, gallium arsenide, germanium rather than through vacuum tubes.
- It do not involve any moving parts or magnetic materials.
- RAM is a solid state device that consists of microchips that store data on non-moving components, providing for fast retrieval of that data.
- Transistors are the most important solid state devices. The transistors contain two p–n junctions, have three contacts or terminals.
- They require the action of perpendicular electrical fields, their behavior is more difficult to understand than that of diodes.
- The different types of transistors are: bipolar junction transistor (BJT) where the current is amplified, while in the field effect transistor (FET) a voltage controls a current.
- In a solid-state component, the current is confined to solid elements and compounds engineered specifically to switch and amplify it.
- Current flows in two forms: as negatively charged electrons, and as positively charged electron deficiencies called holes.
- In some semiconductors, the current consists mostly of electrons; in other semiconductors, it consists mostly of holes. Both the electron and the hole are called charge carriers.

Hard Drives

- A hard disk drive is a non-volatile memory hardware device that permanently stores and retrieves data on a computer.
- A hard drive is a secondary storage device that consists of one or more platters to which data is written using a magnetic head, all inside of an air-sealed casing.

- Internal hard disks reside in a drive bay, connect to the motherboard using an ATA, SCSI, or SATA cable, and are powered by a connection to the power supply unit.

External Hard Drives

- An external hard drive is a portable storage device that can be attached to a computer through a USB or FireWire connection, or wirelessly.
- External hard drives typically have high storage capacities and are often used to back up computers or serve as a network drive.

Optical Drives

- An Optical Drive refers to a computer system that allows users to use DVDs, CDs and Blu-ray optical drives.
- The drive contains some lenses that project electromagnetic waves that are responsible for reading and writing data on optical discs.
- An optical disk drive uses a laser to read and write data. A laser in this context means an electromagnetic wave with a very specific wavelength within or near the visible light spectrum.
- An optical drive that works with all types of discs will have two separate lenses: one for CD/DVD and one for Blu-ray.
- An optical drive has a rotational mechanism to spin the disc. Optical drives were originally designed to work at a constant linear velocity (CLV) (i.e.) the disc spins at varying speeds depending on where the laser beam is reading, so the spiral groove of the disc passes by the laser at a constant speed.
- An optical drive also needs a loading mechanism: A **tray-loading mechanism**, where the disc is placed onto a motorized tray, which moves in and out of the computer case and **slot-loading mechanism**, where the disc is slid into a slot and motorized rollers are used to move the disc in and out.

Tape disks

- A tape drive is a device that stores computer data on magnetic tape, especially for backup and archiving purposes.

- Tape drives work either by using a traditional helical scan where the recording and playback heads touch the tape, or linear tape technology, where the heads never actually touch the tape.
- Drives can be rewinding, where the device issues a rewind command at the end of a session, or non-rewinding.
- Rewinding devices are most commonly used when a tape is to be unmounted at the end of a session after batch processing of large amounts of data.
- Non-rewinding devices are useful for incremental backups and other applications where new files are added to the end of the previous session's files.
- The different types of tapes are audio, video and data storage tape.

Redundant Array of Inexpensive Disks (RAID) Storage

- RAID is a way of storing the same data in different places on multiple hard disks to protect data in the case of a drive failure.
- RAID works by placing data on multiple disks and allowing input/output (I/O) operations to overlap in a balanced way, improving performance. Because the use of multiple disks increases the mean time between failures (MTBF), storing data redundantly also increases fault tolerance.
- A RAID controller can be used as a level of abstraction between the OS and the physical disks, presenting groups of disks as logical units. Using a RAID controller can improve performance and help protect data in case of a crash.

- Levels in RAID:

1. RAID 0 (Disk striping):

RAID 0 splits data across any number of disks allowing higher data throughput. An individual file is read from multiple disks giving it access to the speed and capacity of all of them. This RAID level is often referred to as striping and has the benefit of increased performance.

2. RAID 1 (Disk Mirroring):

RAID 1 writes and reads identical data to pairs of drives. This process is often called data mirroring and its primary function is to provide redundancy. If any of the disks in the array fails, the system can still access data from the remaining disk(s).

3. RAID 5 (Striping with parity):

RAID 5 stripes data blocks across multiple disks like RAID 0, however, it also stores parity information (Small amount of data that can accurately describe larger amounts of data) which is used to recover the data in case of disk failure. This level offers both speed (data is accessed from multiple disks) and redundancy as parity data is stored across all of the disks.

4. RAID 6 (Striping with double parity):

Raid 6 is similar to RAID 5, however, it provides increased reliability as it stores an extra parity block. That effectively means that it is possible for two drives to fail at once without breaking the array.

5. RAID 10 (Striping + Mirroring):

RAID 10 combines the mirroring of RAID 1 with the striping of RAID 0. Or in other words, it combines the redundancy of RAID 1 with the increased performance of RAID 0. It is best suitable for environments where both high performance and security is required.

Universal Serial Bus (USB) Devices

- USB is a system for connecting a wide range of peripherals to a computer, including pointing devices, displays, and data storage and communications products.
- The Universal Serial Bus is a network of attachments connected to the host computer.
- These attachments come in two types known as **Functions and Hubs**.
- Functions are the peripherals such as mice, printers, etc.
- Hubs basically act like a double adapter does on a power-point, converting one socket, called a port, into multiple ports.
- Hubs and functions are collectively called devices.
- When a device is attached to the USB system, it gets assigned a number called its address. The address is uniquely used by that device while it is connected.
- Each device also contains a number of endpoints, which are a collection of sources and destinations for communications between the host and the device.
- The combination of the address, endpoint number and direction are what is used by the host and software to determine along which pipe data is travelling.

Flash Drives

- A flash drive stores data using flash memory. Flash memory uses an electrically erasable programmable read-only (EEPROM) format to store and retrieve data.
- Flash drives are non-volatile, which means they do not need a battery backup.
- Most computers come equipped with USB ports, which detect inserted flash drives and install the necessary drivers to make the data retrievable.
- Computer users can store and retrieve data once the operating system has detected a connection to the USB port.
- Flash drives have a USB mass storage device classification, which means they do not require additional drivers.
- The computer's operating system recognizes a block-structured logical unit, which means it can use any file system or block addressing system to read the information on the flash drive.
- A flash drive enters emulation mode, or acts a hard drive, once it has connected to the USB port. This makes it easier to transfer data between the flash drive and the computer.
- Flash memory is known as a solid state storage device, meaning there are no moving parts — everything is electronic instead of mechanical.

INPUT AND OUTPUT DEVICES

The common input and output devices are discussed here:

Input Devices

Keyboard

- A keyboard has its own processor and circuitry that carries information to and from that processor.
- A large part of this circuitry makes up the **key matrix which is arranged in rows and columns.**
- The key matrix is a grid of circuits underneath the keys.
- In all keyboards each circuit is broken at a point below each key. When a key is presses, it presses a switch, completing the circuit and allowing a tiny amount of current to flow through.

- The mechanical action of the switch causes some vibration, called **bounce**, which the processor filters out.
- If the key is pressed and held continuously, the processor recognizes it as the equivalent of pressing a key repeatedly.
- Another type of keyboard has three layers: top plasticized layer with key positions marked on the top surface and conducting traces on another side; middle layer made of rubber with hole for key positions; bottom metallic layer with raised bumps for key positions.
- When a key is pressed the trace underneath the top layer comes in contact with the bump in the last layer, thus completing an electrical circuit. The current flow is sensed by the microcontroller.

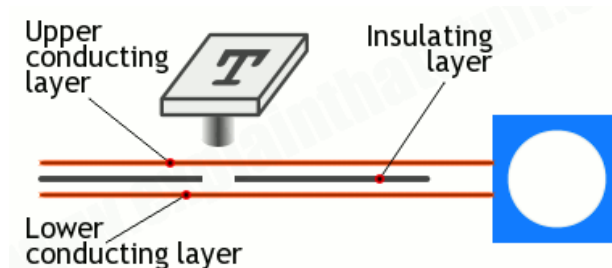


Fig 4.24: Layers in keyboard

Mouse

- A computer mouse is a hand-held pointing device that detects two-dimensional motion relative to a surface.
- This motion is typically translated into the motion of a pointer on a display, which allows a smooth control of the graphical user interface.
- There are two main kinds of mice: rolling rubber ball mouse or optical mouse.
- As the mouse is moved, the ball rolls under its own weight and pushes against two plastic rollers linked to thin wheels.
- One of the wheels detects movements in an up-and-down direction (y-axis) and the other detects side-to-side movements (x-axis).
- If the mouse is moved straight up, only the y-axis wheel turns. If the mouse is moved to the right, only the x-axis wheel turns.

- The optical mouse shines a bright light down onto the desk from an LED mounted on the bottom of the mouse.
- The light bounces straight back up off the desk into a photocell also mounted under the mouse, a short distance from the LED.
- The photocell has a lens in front of it that magnifies the reflected light, so the mouse can respond more precisely to your hand movements.
- As the mouse is pushed, the pattern of reflected light changes, and the chip inside the mouse uses this to figure out the motion.

Trackball, Joystick and Touch pad

- A trackball can also be used as an alternative to a mouse. This device also has buttons similar to those on a mouse.
- It holds a large moving ball on the top. The body of the trackball is not moved. The ball is rolled with fingers. The position of the cursor on the screen is controlled by rotating the ball.
- The main benefit of the trackball over a mouse is that it takes less space to move. The trackball is often included in laptop computers. The standard desktop computer also uses a trackball operated as a separate input device.
- A touchpad is a small, plane surface over which the user moves his finger. The user controls the movement of the cursor on the screen by moving his fingers on the touchpad. It is also known as a trackpad.
- A touchpad also has one or more buttons near it. These button work like mouse buttons. Touchpads are commonly used with notebook computers.
- A joystick consists of a base and a stick. The stick can be moved in several directions to shift an object anywhere on the computer screen.
- A joystick can perform a similar function to a mouse or trackball. It is often considered less comfortable and efficient. The most common use of a joystick is for playing computer games.

Scanners

- Scanners operate by shining light at the object or document being digitized and directing the reflected light onto a photosensitive element.
- In most scanners, the sensing medium is an electronic, light-sensing integrated circuit known as a charged coupled device (CCD).
- Light-sensitive photosites arrayed along the CCD convert levels of brightness into electronic signals that are then processed into a digital image.
- A scanner consists of a flat transparent glass bed under which the CCD sensors, lamp, lenses, filters and also mirrors are fixed.
- The document has to be placed on the glass bed. There will also be a cover to close the scanner.
- The lamp brightens up the text to be scanned. Most scanners use a cold cathode fluorescent lamp (CCFL).
- A stepper motor under the scanner moves the scanner head from one end to the other. The movement will be slow and is controlled by a belt.
- The scanner head consists of the mirrors, lens, CCD sensors and also the filter. The scan head moves parallel to the glass bed and that too in a constant path.
- As the scan head moves under the glass bed, the light from the lamp hits the document and is reflected back with the help of mirrors angled to one another.
- According to the design of the device there may be either 2-way mirrors or 3-way mirrors.
- The mirrors will be angled in such a way that the reflected image will be hitting a smaller surface.
- In the end, the image will reach a lens which passes it through a filter and causes the image to be focussed on CCD sensors.
- The CCD sensors convert the light to electrical signals according to its intensity.
- The electrical signals will be converted into image format inside a computer.

Output Devices Video

Displays

- The CRT monitors were fundamental output display device.
- The CRT or cathode ray tube, is the picture tube of a monitor.
- The back of the tube has a negatively charged cathode.
- The electron gun shoots electrons down the tube and onto a charged screen.
- The screen is coated with a pattern of dots using phosphor that glow when struck by the electron stream.
- The image on the monitor screen is usually made up from at least tens of thousands of such tiny dots glowing on command from the computer.
- The closer together the pixels are, the sharper the image on screen.
- The distance between pixels on a computer monitor screen is called its dot pitch and is measured in millimeters. Most monitors have a dot pitch of 0.28 mm or less.
- There are two electromagnets around the collar of the tube which deflect the electron beam.
- The beam scans across the top of the monitor from left to right, is then blanked and moved back to the left-hand side slightly below the previous trace (on the next scan line), scans across the second line and so on until the bottom right of the screen is reached.
- The beam is again blanked, and moved back to the top left to start again.
- This process draws a complete picture, typically 50 to 100 times a second.
- The number of times in one second that the electron gun redraws the entire image is called the refresh rate and is measured in hertz (cycles per second).
- It is common, particularly in lower priced equipment, for all the odd-numbered lines of an image to be traced, and then all the even-numbered lines; the circuitry of such an interlaced display need to be have only half the speed of a non-interlaced display.
- An interlaced display, particularly at a relatively low refresh rate, can appear to some observers to flicker, and may cause eye strain and nausea.

- The intensity or strength of the electron beam is controlled by setting the voltage levels.
- The number of electrons that hits the screen determines the light emitted by the screen. When the voltage is varied in the electron gun, the brightness of the display also varies.
- The focusing hardware focuses the beam at all positions on the screen.
- The deflection of electron beam is controlled by electric or magnetic fields.
- Two pairs of coils mounted on the CRT to produce the necessary deflection.
- The coils are placed in such a way that, the magnetic field produced by them results in traverse deflection force that is perpendicular to the magnetic field and electron beam.

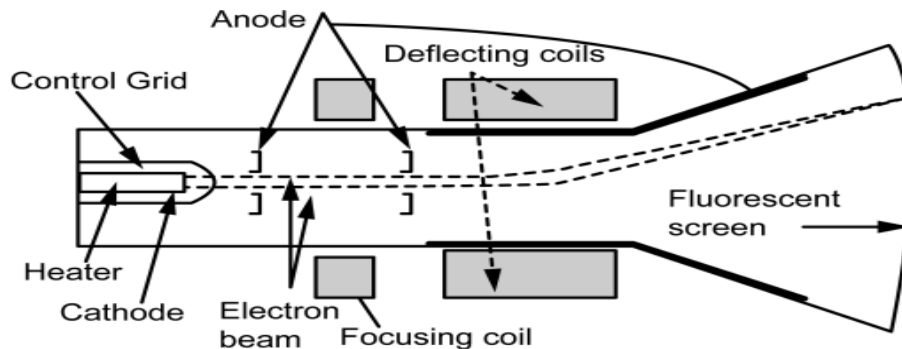


Fig 4.25: CRT Monitor

- An LED screen is an LCD screen, but instead of having a normal CCFL backlight, it uses light-emitting diodes (LEDs) as a source of light behind the screen.
- An LED is more energy efficient and a lot smaller than a CCFL, enabling a thinner television screen.

Printers

- A printer is an electromechanical device which converts the text and graphical documents from electronic form to the physical form.
- They are the external peripheral devices which are connected with the computers or laptops through a cable or wirelessly to receive input data and print them on the papers.
- Quality of printers is identified by its features like color quality, speed of printing, resolution etc. Modern printers come with multipurpose functions i.e. they are combination of printer, scanner, photocopier, fax, etc.
- Broadly printers are categorized as impact and non impact printers.

Daisy Wheel Printers

- Daisy wheel printers print only characters and symbols and cannot print graphics. They are generally slow with a printing speed of about 10 to 75 characters per second.
- A circular printing element is the heart of these printers that contains all text, numeric characters and symbols mould on each petal on the circumference of the circle.
- The printing element rotates rapidly with the help of a servo motor and pauses to allow the printing hammer to strike the character against the paper.

Dot Matrix Printers

- It is a popular computer printer that prints text and graphics on the paper by using tiny dots to form the desired shapes.
- It uses an array of metal pins known as printhead to strike an inked printer ribbon and produce dots on the paper.
- These combinations of dots form the desired shape on the paper.
- The key component in the dot matrix printer is the 'printhead' which is about one inch long and contains a number of tiny pins aligned in a column varying from 9 to 24.
- The printhead is driven by several hammers which force each pin to make contact with the paper at the certain time. These hammers are pulled by small electromagnet which is energized at a specific time depending on the character to be printed.
- The timings of the signals sent to the solenoids are programmed in the printer for each character.

Inkjet printers

- Inkjet printers are most popular printers for home and small scale offices as they have a reasonable cost and a good quality of printing as well.
- An inkjet printer is made of the following parts:
 - i) Printhead – It is the heart of the printer which holds a series of nozzles which spray the ink drops over the paper.
 - ii) Ink cartridge – It is the part that contains the ink for printing. Generally monochrome (black & white) printers contain a black colored ink cartridge and a color printer

contains two cartridges – one with black ink and other with primary colors (cyan, magenta and yellow).

- iii) Stepper motor – It is housed in the printer to move the printerhead and ink cartridges back and forth across the paper.
- iv) Stabilizer bar – A stabilizer bar is used in printer to ensure the movement of printhead is précised and controlled over the paper.
- v) Belt – A belt is used to attach the printhead with the stepper motor.
- vi) Paper Tray – It is the place where papers are placed to be printed.
- vii) Rollers – Printers have a set of rollers that helps to pull paper from the tray for printing purpose.
- viii) Paper tray stepper motor- another stepper motor is used to rotate the rollers in order to pull the paper in the printer.
- ix) Control Circuitry – The control circuit takes the input from the computer and by decoding the input controls all mechanical operation of the printer.

Laser Printers

- Laser printers are the most popular printers that are mainly used for large scale qualitative printing.
- They are among the most popularly used fastest printers available in the market.
- A laser printer uses a slight different approach for printing. It does not use ink like inkjet printers, instead it uses a very fine powder known as Toner.
- The control circuitry is the part of the printer that talks with the computer and receives the printing data.
- A Raster Image Processor (RIP) converts the text and images in to a virtual matrix of dots.
- The photoconducting drum which is the key component of the laser printer has a special coating which receives the positive and negative charge from a charging roller.
- A rapidly switching laser beam scans the charged drum line by line. When the beam flashes on, it reverses the charge of tiny spots on the drum, respecting to the dots that are to be printed black.

- As soon the laser scans a line, a stepper motor moves the drum in order to scan the next line by the laser.
- A developer roller plays the vital role to paste the tonner on the paper. It is coated with charged tonner particles.
- As the drum touches the developer roller, the charged tonner particles cling to the discharged areas of the drum, reproducing your images and text reversely.
- Meanwhile a paper is drawn from the paper tray with help of a belt. As the paper passes through a charging wire it applies a charge on it opposite to the toner's charge.
- When the paper meets the drum, due to the opposite charge between the paper and toner particles, the toner particles are transferred to the paper.
- A cleaning blade then cleans the drum and the whole process runs smoothly continuously.
- Finally paper passes through the fuser which is a heat and presser roller, melts the toner and fixes on the paper perfectly.

UNIT - V

ADVANCED COMPUTER ARCHITECTURE

PARALLEL PROCESSING ARCHITECTURES

Parallel computing architectures breaks the job into discrete parts that can be executed concurrently. Each part is further broken down to a series of instructions. Instructions from each part execute simultaneously on different CPUs. Parallel systems deal with the simultaneous use of multiple computer resources that can include a single computer with multiple processors, a number of computers connected by a network to form a parallel processing cluster or a combination of both. Parallel systems are more difficult to program than computers with a single processor because the architecture of parallel computers varies accordingly and the processes of multiple CPUs must be coordinated and synchronized. The crux of parallel processing are the CPUs.

Flynn's taxonomy is a specific classification of parallel computer architectures that are based on the number of concurrent instruction (single or multiple) and data streams (single or multiple) available in the architecture.

Parallelism in computer architecture is explained using Flynn's taxonomy. This classification is based on the number of instruction and data streams used in the architecture. The machine structure is explained using streams which are sequence of items. The four categories in Flynn's taxonomy based on the number of instruction streams and data streams are the following:

- (SISD) single instruction, single data
- (MISD) multiple instruction, single data
- (SIMD) single instruction, multiple data
- (MIMD) multiple instruction, multiple data

SISD (Single Instruction, Single Data stream)

- Single Instruction, Single Data (SISD) refers to an Instruction Set Architecture in which a single processor (one CPU) executes exactly one instruction stream at a time.
- It also fetches or stores one item of data at a time to operate on data stored in a single memory unit.

- Most of the CPU design is based on the von Neumann architecture and the follow SISD.
- The SISD model is a non-pipelined architecture with general-purpose registers, Program Counter (PC), the Instruction Register (IR), Memory Address Registers (MAR) and Memory Data Registers (MDR).

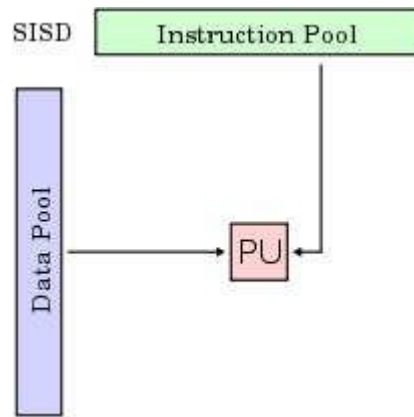


Fig 5.1: Single Instruction, Single Data Stream

SIMD (Single Instruction, Multiple Data streams)

- Single Instruction, Multiple Data (SIMD) is an Instruction Set Architecture that have a single control unit (CU) and more than one processing unit (PU) that operates like a von Neumann machine by executing a single instruction stream over PUs, handled through the CU.
- The CU generates the control signals for all of the PUs and by which executes the same operation on different data streams.
- The SIMD architecture is capable of achieving data level parallelism.

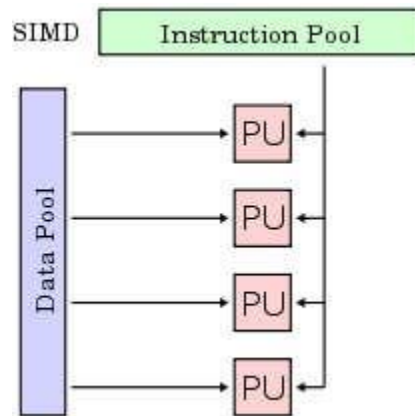


Fig 5.2:Single Instruction, Multiple Data streams

MISD (Multiple Instruction, Single Data stream)

- Multiple Instruction, Single Data (MISD) is an Instruction Set Architecture for parallel computing where many functional units perform different operations by executing different instructions on the same data set.
- This type of architecture is common mainly in the fault-tolerant computers executing the same instructions redundantly in order to detect and mask errors.

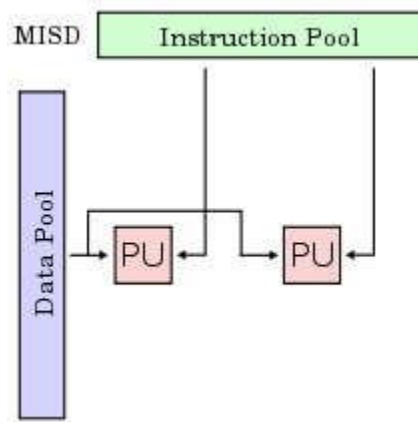
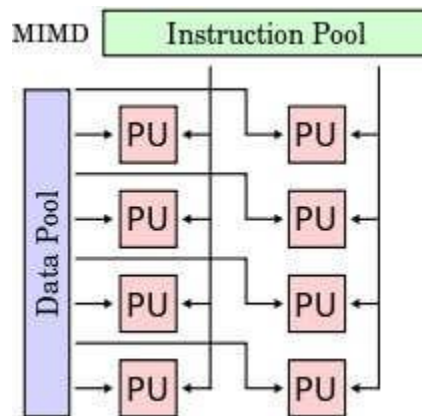


Fig 5.3:Multiple Instruction, Single Data stream

MIMD (Multiple Instruction, Multiple Data streams)

- Multiple Instruction stream, Multiple Data stream (MIMD) is an Instruction Set Architecture for parallel computing that is typical of the computers with multiprocessors.
- Using the MIMD, each processor in a multiprocessor system can execute asynchronously different set of the instructions independently on the different set of data units.
- The MIMD based computer systems can use the shared memory in a memory pool or work using distributed memory across heterogeneous network computers in a distributed environment.
- The MIMD architectures is primarily used in a number of application areas such as computer-aided design/computer-aided manufacturing, simulation, modelling, communication switches etc.

**Fig 5.4: Multiple Instruction, Multiple Data streams**

	Single	Multiple
Single	SISD Von Neumann Single computer	MISD May be pipelined computers
Multiple	SIMD Vector processors Fine grained dataParallel computers	MIMD Multi computers Multiprocessors

Fig 5.5: Comparison of Flynn's taxonomy

Challenges in Parallelism

The following are the design challenges in parallelism:

- Available parallelism.
- Load balance: Some processors work while others wait due to insufficient parallelism or unequal size tasks.
- Extra work.
- Managing parallelism
- Redundant computation
- Communication

HARDWARE MULTITHREADING

Multithreading enables the processing of multiple threads at one time, rather than multiple processes. Since threads are smaller, more basic instructions than processes, multithreading may occur within processes. Threads are instruction stream with state (registers and memory). The register state is also called thread context. Threads could be part of the same process or from different programs. Threads in the same program share the same address space and hence consume fewer resources.

The terms multithreading, multiprocessing and multitasking are used interchangeably. But each has its unique meaning:

- **Multitasking:** It is the process of executing multiple tasks simultaneously. In multitasking, when a new thread needs to be executed, old thread's context in hardware written back to memory and new thread's context loaded.
- **Multiprocessing:** It is using two or more CPUs within a single computer system.
- **Multithreading:** It is executing several parts of a program in parallel by dividing the specific operations within a single application into individual threads.

Granularity: The threads are categorized based on the amount of work done by the thread. This is known as granularity. When the hardware executes from the hardware contexts determines the granularity of multithreading.

Hardware vs Software multithreading

Software Multithreading	Hardware Multithreading
Execution of concurrent threads is supported by OS.	Execution of concurrent threads is supported by CPU.
Large number of threads can be span.	Very limited number of threads can span.
Context switching is heavy. It involves more operations.	Light/ immediate context switching with limited operations.

Hardware multithreading is having multiple threads contexts to span in same processor. This is supported by the CPU.

The following are the objectives of hardware multithreading:

- To tolerate latency of memory operations, dependent instructions, branch resolution by utilizing processing resources more efficiently. When one thread encounters a long-latency operation, the processor can execute a useful operation from another thread.
- To improve system throughput v By exploiting thread-level parallelism by improving superscalar processor utilization
- To reduce context switch penalty

Advantages of hardware multithreading:

- Latency tolerance
- Better hardware utilization
- Reduced context switch penalty

Cost of hardware multithreading:

- Requires multiple thread contexts to be implemented in hardware.
- Usually reduced single-thread performance
- Resource sharing, contention
- Switching penalty (can be reduced with additional hardware)

Types of hardware multithreading

The hardware multithreading is classified based on the granularity of the threads as:

- Fine grained
- Coarse grained
- Simultaneous

Fine Grained Multithreading

- Here, the CPU switch to another thread at every cycle such that no two instructions from the thread are in the pipeline at the same time. Hence it is also known as **interleaved multithreading**.

Fine grained multithreading is a mechanism in which switching among threads happen despite the cache miss or stall caused by the thread instruction.

- The threads are executed in a round-robin fashion in consecutive cycles.
- The CPU checks every cycle if the current thread is stalled or not.
- If stalled, a hardware scheduler will change execution to another thread that is ready to run.
- Since the hardware is checking every cycle for stalls, all stall types can be dealt with, even single cycle stalls.
- This improves pipeline utilization by taking advantage of multiple threads
- It tolerates the control and data dependency latencies by overlapping the latency with useful work from other threads
- Fine-grained parallelism is best exploited in architectures which support fast communication.
- Shared memory architecture which has a low communication overhead is most suitable for fine-grained parallelism.
- This requires more threads to keep the CPU busy.

Advantages:

- No need for dependency checking between instructions since only one instruction in pipeline from a single thread.

- No need for branch prediction logic.
- The bubble cycles used for executing useful instructions from different threads.
- Improved system throughput, latency tolerance, utilization.

Disadvantages:

- Extra hardware complexity because of implementation of multiple hardware contexts and thread selection logic.
- Reduced single thread performance as one instruction fetched every N cycles.
- Resource contention between threads in caches and memory.
- Dependency checking logic between threads remains.

Coarse grained multithreading

- In this type, the instructions of other threads are executed successively until an event in current execution thread cause latency. This delay event induces a context switch.
- When a thread is stalled due to some event, the CPU switch to a different hardware context. This is known as Switch-on-event multithreading or **blocked multithreading**.

Coarse grained multithreading is a mechanism in which the switch only happens when the thread in execution causes a stall, thus wasting a clock cycle.

- This is less efficient than fine grained multithreading but requires only few threads to improve CPU utilization.
- The events that causes latency or stalls are: Cache misses, Synchronization events and floating point operations.
- Resource sharing in space and time always requires fairness considerations. This is implemented by considering how much progress each thread makes.
- The time allocated to each thread affects both fairness and system throughput. The allocation strategies depends on the answers to the following questions:
 - When do we switch?
 - For how long do we switch?
 - When do we switch back?

- How does the hardware scheduler interact with the software scheduler for fairness?
- What is the switching overhead vs. benefit?
- Where do we store the contexts?
- A trade off must be done between fairness and system throughput: Switch not only on miss, but also on data return.
- This has a severe problem because switching has performance overhead as it requires flushing of pipeline and window; reduced locality and increased resource contention.
- One possible solution is to estimate the slowdown of each thread compared to when run alone. Then enforce switching when slowdowns become significantly unbalanced.

Advantages:

- Simpler to implement, can eliminate dependency checking and branch prediction logic completely
- Switching need not have any performance overhead.
- Higher performance overhead with deep pipelines and large windows

Disadvantages

- Low single thread performance: each thread gets 1/Nth of the bandwidth of the pipeline

Simultaneous Multithreading (SMT)

- Here instructions can be issued from multiple threads in any given cycle.
- Instructions are simultaneously issued from multiple threads to the execution units of a superscalar processor. Thus, the wide superscalar instruction issue is combined with the multiple-context approach.
- In fine-grained and coarse-grained architectures, multithreading can start execution of instructions from only a single thread at a given cycle.
- Execution unit or pipeline stage utilization can be low if there are not enough instructions from a thread to dispatch in one cycle

- Unused instruction slots, which arise from latencies during the pipelined execution of single-threaded programs by a microprocessor, are filled by instructions of other threads within a multithreaded processor.
- The execution units are multiplexed among those thread contexts that are loaded in the register sets.
- Underutilization of a superscalar processor due to missing instruction-level parallelism can be overcome by simultaneous multithreading, where a processor can issue multiple instructions from multiple threads in each cycle.
- Simultaneous multithreaded processors combine the multithreading technique with a wide-issue superscalar processor to utilize a larger part of the issue bandwidth by issuing instructions from different threads simultaneously.

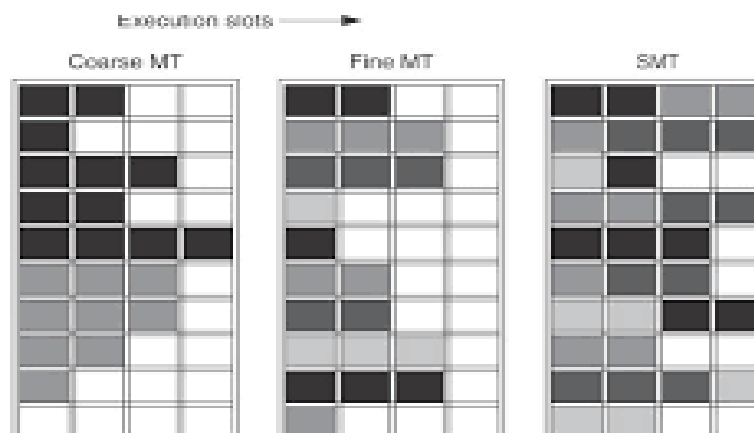


Fig 5.6: Hardware multithreading

MULTICORE AND SHARED MEMORY MULTIPROCESSORS

A multi-core processor is a single computing component with two or more independent processing units called cores, which read and execute program instructions. A shared-memory multiprocessor is a computer system composed of multiple independent processors that execute different instruction streams.

- Multi-core is usually the term used to describe two or more CPUs working together on the same chip. It is a type of architecture where a single physical processor contains the core logic of two or more processors.

- Shared Memory Processor (SMP) follows multiple-instruction multiple-data (MIMD) architecture.
- The processors share a common memory address space and communicate with each other via memory. All the processors will have dedicated cache memory.
- In a multiprocessor system all processes on the various CPUs share a unique logical address space, which is mapped on a physical memory that can be distributed among the processors.
- Each process can read and write a data item simply using load and store operations, and process communication is through shared memory.
- It is the hardware that makes all CPUs access and use the same main memory.
- Since all CPUs share the address space, only a single instance of the operating system is required.
- When a process terminates or goes into a wait state for whichever reason, the O.S. can look in the process table for another process to be dispatched to the idle CPU.
- On the contrary, in systems with no shared memory, each CPU must have its own copy of the operating system, and processes can only communicate through message passing.
- The basic issue in shared memory multiprocessor systems is memory itself, since the larger the number of processors involved, the more difficult to work on memory efficiently.
- All modern OS support symmetric multiprocessing, with a scheduler running on every processor the ready to run processes can be inserted into a single queue, that can be accessed by every scheduler, alternatively there can be a “ready to run” queue for each processor.
- When a scheduler is activated in a processor, it chooses one of the ready to run processes and dispatches it on its processor.

Load Balancing:

- A distinct feature in multiprocessor systems is load balancing.

- It is useless having many CPUs in a system, if processes are not distributed evenly among the cores.
- With a single ready-to-run queue, load balancing is usually automatic: if a processor is idle, its scheduler will pick a process from the shared queue and will start it on that processor.
- Modern OSs designed for SMP often have a separate queue for each processor to avoid the problems associated with a single queue.
- There is an explicit mechanism for load balancing, by which a process on the wait list of an overloaded processor is moved to the queue of another, less loaded processor.

Types of shared memory multiprocessors

There are three types of shared memory multiprocessors:

- i) Uniform Memory Access (UMA)
- ii) Non Uniform Memory Access (NUMA)
- iii) Cache Only Memory Access (COMA)

i) Uniform Memory Access (UMA)

- Here, all the processors share the physical memory in a centralized manner with equal access time to all the memory words.
- Each processor may have a private cache memory. Same rule is followed for peripheral devices.
- When all the processors have equal access to all the peripheral devices, the system is called a **symmetric multiprocessor**.
- When only one or a few processors can access the peripheral devices, the system is called an **asymmetric multiprocessor**.
- When a CPU wants to access a memory location, it checks if the bus is free, then it sends the request to the memory interface module and waits for the requested data to be available on the bus.
- Multicore processors are small UMA multiprocessor systems, where the first shared cache is actually the communication channel.

- Shared memory can quickly become a bottleneck for system performances, since all processors must synchronize on the single bus and memory access.

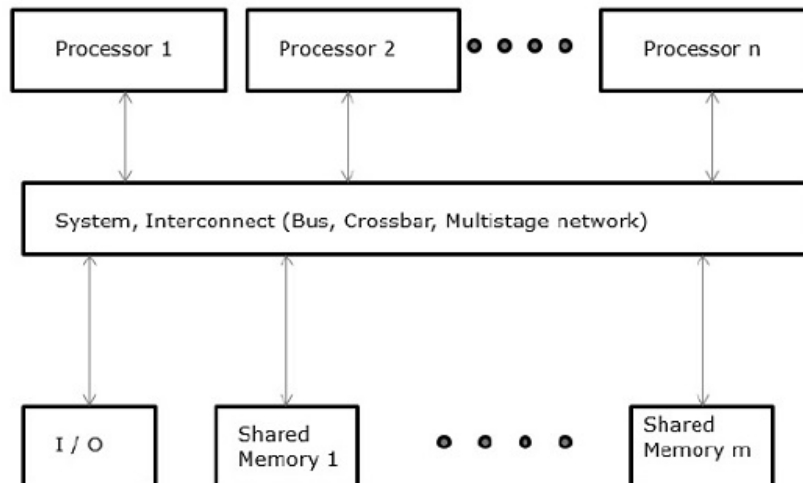


Fig 5.7: Uniform memory access model

ii) Non-uniform Memory Access (NUMA)

- In NUMA multiprocessor model, the access time varies with the location of the memory word.
- Here, the shared memory is physically distributed among all the processors, called local memories.
- The collection of all local memories forms a global address space which can be accessed by all the processors.
- NUMA systems also share CPUs and the address space, but each processor has a local memory, visible to all other processors.
- In NUMA systems access to local memory blocks is quicker than access to remote memory blocks.
- Programs written for UMA systems run with no change in NUMA ones, possibly with different performances because of slower access times to remote memory blocks.
- Single bus UMA systems are limited in the number of processors, and costly hardware is necessary to connect more processors.

- Current technology prevents building UMA systems with more than 256 processors.
- To build larger processors, a compromise is mandatory: not all memory blocks can have the same access time with respect to each CPU.
- Since all NUMA systems have a single logical address space shared by all CPUs, while physical memory is distributed among processors, there are two types of memories: local and remote memory.

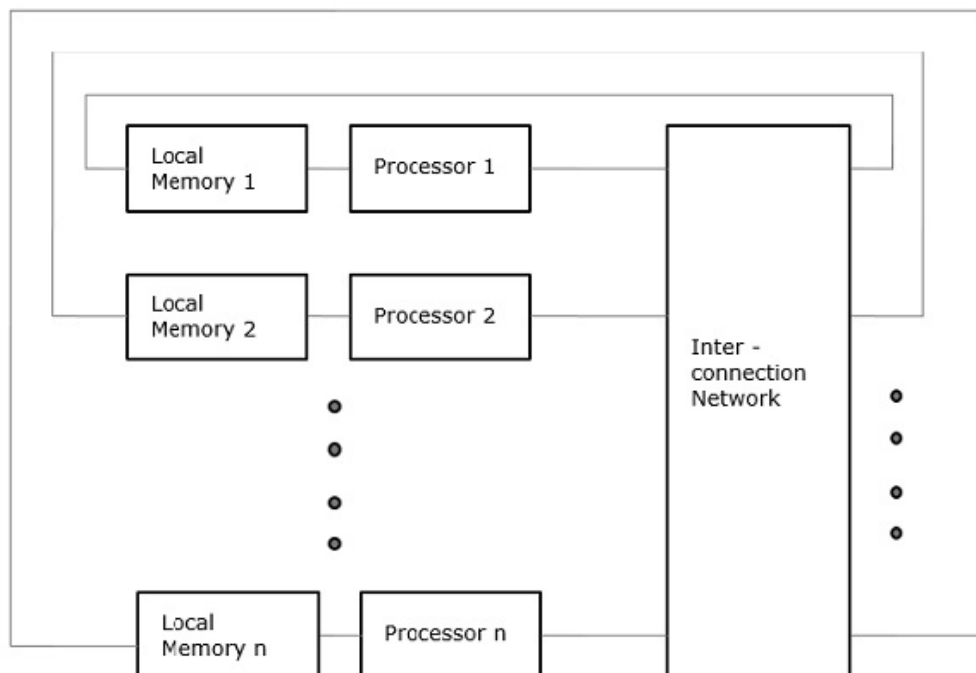


Fig 5.8:Non-uniform Memory Access model

- There are two types of NUMA systems: Non-Caching NUMA (NC-NUMA) Cache-Coherent NUMA (CC-NUMA).
 - **Non-Caching NUMA (NC-NUMA):**
 - In a NC-NUMA system, processors have no local cache. Each memory access is managed with a modified MMU, which controls if the request is for a local or for a remote block; in the latter case, the request is forwarded to the node containing the requested data.

- Obviously, programs using remote data will run much slower than what they would, if the data were stored in the local memory. In NC-NUMA systems there is no cache coherency problem, because there is no caching at all: each memory item is in a single location.
- Remote memory access is however very inefficient. For this reason, NC-NUMA systems can resort to special software that relocates memory pages from one block to another, just to maximise performances.

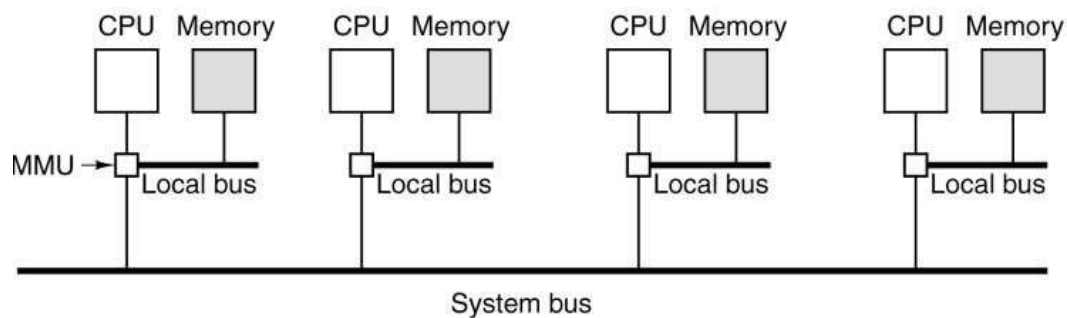


Fig 5.9: Non-Caching NUMA

• **Cache-Coherent NUMA (CC-NUMA):**

- Caching can alleviate the problem due to remote data access, but brings the cache coherency issue.
- A method to enforce coherency is obviously bus snooping, but this technique gets too expensive beyond a certain number of CPUs, and it is much too difficult to implement in systems that do not rely on bus-based interconnections.
- The common approach in CC-NUMA systems with many CPUs to enforce cache coherency is the directory-based protocol.
- The basic idea is to associate each node in the system with a directory for its RAM blocks: a database stating in which cache is located a block, and what is its state.
- When a block of memory is addressed, the directory in the node where the block is located is queried, to know if the block is in any cache and, if so, if it has been changed respect to the copy in RAM.

- Since a directory is queried at each access by an instruction to the corresponding memory block, it must be implemented with very quick hardware, as an instance with an associative cache, or at least with static RAM.

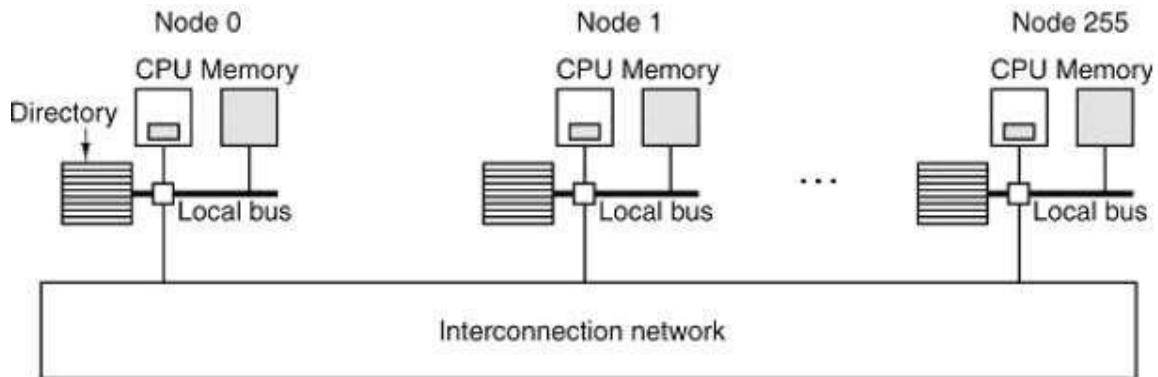


Fig 5.10:Cache-Coherent NUMA

iii) Cache Only Memory Access (COMA)

- The COMA model is a special case of the NUMA model. Here, all the distributed main memories are converted to cache memories.
- In a monoprocessor architecture and in shared memory architectures each block and each line are located in a single, precise position of the logical address space, and have therefore an address called home address.
- When a processor accesses a data item, its logical address is translated into the physical address, and the content of the memory location containing the data is copied into the cache of the processor, where it can be read and/or modified.
- In the last case, the copy in RAM will be eventually overwritten with the updated copy present in the cache of the processor that modified it.
- This property turns the relationship between processors and memory into a critical one, both in UMA and in NUMA systems:
- In NUMA systems, distributed memory can generate a high number of messages to move data from one CPU to another, and to maintain coherency in home address values. Remote memory references are much slower than local memory ones.

- In CC-NUMA systems, this effect is partially hidden by the caches.
- In UMA systems, centralized memory causes a bottleneck, and limit its the interconnection between CPU and memory, and its scalability.
- In COMA, there is no longer a home address, and the entire physical address space is considered a huge, single cache.
- Data can migrate within the whole system, from a memory bank to another, according to the request of a specific CPU, that requires that data.

GRAPHICS PROCESSING UNITS

A Graphics Processing Unit (GPU) is a single-chip processor primarily used to manage and boost the performance of video and graphics. It is a dedicated parallel processor for accelerating graphical and deeper computations.

GPU is designed to lessen the work of the CPU and produce faster video and graphics. GPU can be thought as an extension of CPU with thousands of cores. A GPU is extensively used in a PC on a video card or motherboard, mobile phones, display adapters, workstations and game consoles. They are mainly used for offloading computation intensive application. This is also known as a **visual processing unit (VPU)**.

Differences between CPU and GPU

GPU	CPU
They facilitate highly parallel operations.	This supports serial execution of programs.
This has more number of cores (in thousands).	This has less number of cores.
They need special faster interfaces to facilitate faster data transfers.	No such special interfaces are required.
They have deeper pipelines.	They have comparatively shallow pipelines.

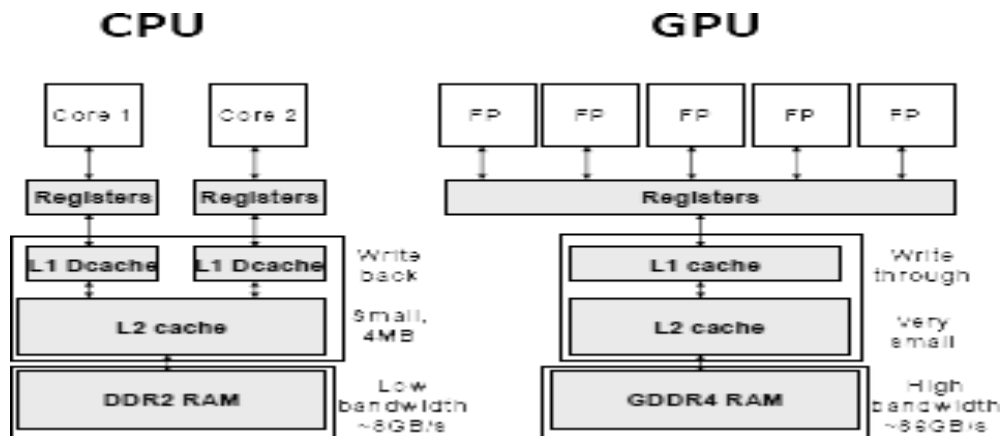


Fig 5.11: CPU vs GPU architecture

GPU features

The following are prominent features of GPU:

- 2-D or 3-D graphics
- Digital output to flat panel display monitors
- Texture mapping
- Application support for high-intensity graphics software such as AutoCAD
- Rendering polygons
- Support for YUV color space
- Hardware overlays
- MPEG decoding

Development of GPU

- The first GPU was developed by NVidia in 1999 and named as GeForce 256.
- This GPU model could process 10 million polygons per second and had more than 22 million transistors.
- This is a single-chip processor with integrated transform, drawing and BitBLT support, lighting effects, triangle setup /clipping and rendering engines.

- The GPU is connected to the CPU and is completely separate from the motherboard.
- The RAM is connected through the Accelerated Graphics Port (AGP) or the PCI express bus.
- Sometimes, GPUs are integrated into the northbridge on the motherboard and use the main memory as a digital storage area, but these GPUs are slower and have poorer performance.
- The accelerated memory in GPU is used for mapping vertices and can also supports programmable shaders implementing textures, mathematical vertices and accurate color formats.
- Applications such as Computer-Aided Design (CAD) can process over 200 billion operations per second and deliver up to 17 million polygons per second.
- The main configurations of GPU processor are: Graphics coprocessor which is independent of CPU and Graphics accelerator that is based on commands from CPU.

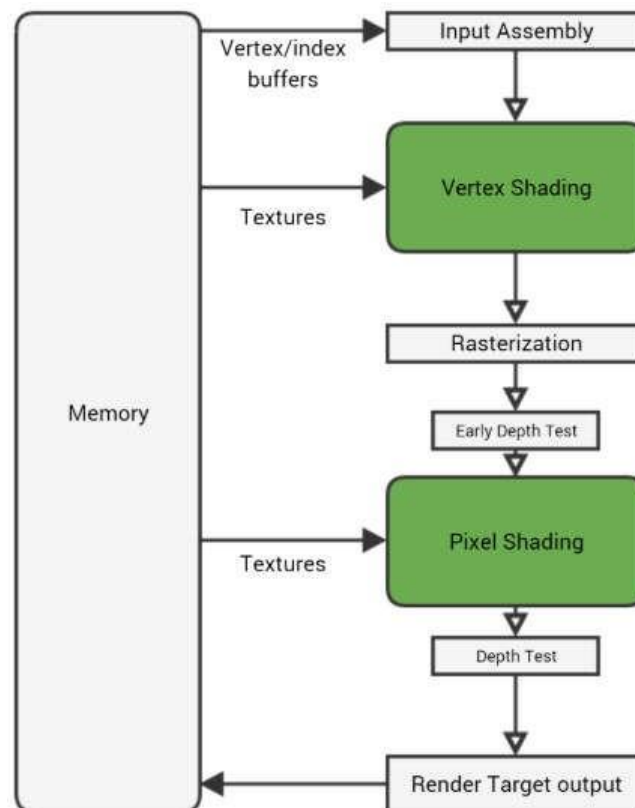


Fig 5.12: GPU Pipeline

Input Assembler stage

- This stage is the communication bridge between the CPU and GPU.
- It receives commands from the CPU and also pulls geometry information from system memory.
- It outputs a stream of vertices in object space with all their associated information.

Vertex Processing

- This processes vertices performing operations like transformation, skinning and lighting.
- A vertex shader takes a single input vertex and produces a single output vertex.

Pixel Processing

- Each pixel provided by triangle setup is fed into pixel processing as a set of attributes which are used to compute the final color for this pixel.
- The computations taking place here include texture mapping and math operations

Output Merger Stage

- The output-merger stage combines various types of output data to generate the final pipeline result.

CLUSTERS AND WAREHOUSE SCALE COMPUTERS

A cluster is a collection of desktop computers or servers connected together by a local area network to act as a single larger computer. A warehouse-scale computer (WSC) is a cluster comprised of tens of thousands of servers

Warehouse-scale computers form the foundation of internet services. The present days WSCs act as one giant machine. The main parts of a WSC are the building with the electrical and cooling infrastructure, the networking equipment and the servers.

WSCs as Servers

The following features of WSCs that makes it work as servers:

- **Cost-performance:** Because of the scalability, the cost-performance becomes very critical. Even small savings can amount to a large amount of money.

- **Energy efficiency:** Since large numbers of systems are clustered, lot of money is invested in power distribution and for heat dissipation. Work done per joule is critical for both WSCs and servers because of the high cost of building the power and mechanical infrastructure for a warehouse of computers and for the monthly utility bills to power servers. If servers are not energy-efficient they will increase
 - cost of electricity
 - cost of infrastructure to provide electricity
 - cost of infrastructure to cool the servers.
- **Dependability via redundancy:** The hardware and software in a WSC must collectively provide at least 99.99% availability, while individual servers are much less reliable. Redundancy is the key to dependability for both WSCs and servers. WSC architects rely on multiple cost-effective servers connected by a low cost network and redundancy managed by software. Multiple WSCs may be needed to handle faults in whole WSCs. Multiple WSCs also reduce latency for services that are widely deployed.
- **Network I/O:** Networking is needed to interface to the public as well as to keep data consistent between multiple WSCs.
- **Interactive and batch-processing workloads:** Search and social networks are interactive and require fast response times. At the same time, indexing, big data analytics etc. create a lot of batch processing workloads also. The WSC workloads must be designed to tolerate large numbers of component faults without affecting the overall performance and availability.

Differences between WSCs and data centers

Data Centers	WSCs
Data centres hosts services for multiple providers.	WSCs are run by only one client.
There will be little commonality between hardware and software.	Homogenous hardware and software management.
Third party software solutions.	In-house middle ware.

WSC are not servers:

The following features of WSCs make them different from servers:

- **Ample parallelism:**
 - Servers need not to worry about the parallelism available in applications to justify the amount of parallel hardware.
 - But in WSCs most jobs are totally independent and exploit request-level parallelism.
 - **Request-Level parallelism (RLP)** is a way of representing tasks which are set of requests which are to be to run in parallel.
 - Interactive internet service applications, the workload consists of independent requests of millions of users.
 - Also, the data of many batch applications can be processed in independent chunks, exploiting data-level parallelism.
- **Operational costs count:**
 - Server architects normally design systems for peak performance within a cost budget.
 - Power concerns are not too much as long as the cooling requirements are maintained. The operational costs are ignored.
 - WSCs, however, have a longer life times and the building, electrical and cooling costs are very high.
 - So, the operational costs cannot be ignored. A
 - If these add up to more than 30% of the costs of a WSC in 10 years.
 - Power consumption is a primary, not secondary constraint when designing the WSC system.
- **Scale and its opportunities and problems:**
 - The WSCs are massive internally, so it gets volume discounts and economy of scale, even if there are not too many WSCs.
 - On the other hand, customized hardware for WSCs can be very expensive, particularly if only small numbers are manufactured.

- The economies of scale lead to cloud computing, since the lower per-unit costs of WSCs lead to lower rental rates.
- Even if a server had a Mean Time To Failure (MTTF) of twenty five years, the WSC architect should design for five server failures per day.

Architecture of WSC

The height of the servers is measured by **rack units**. A typical rack is 42 rack units. But the standard dimension to hold the servers is 48.26 cm.

1 rack unit (U)=1.75 inches or 44.45 mm.

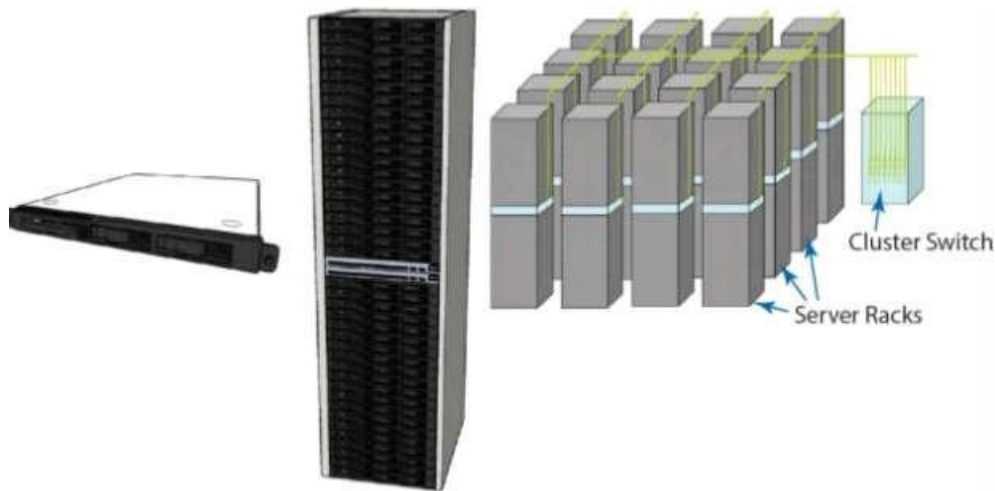


Fig 5.13: Architecture of WSCs

The fig 5.13 shows a WSC system with 1Unit server, 7 inch rack with an Ethernet switch. This figure shows a high end server. But low end servers are of 1U size mounted within a rack and connected with Ethernet switch. These rack level switches use 1 or 10 Gbps links with a number of uplink connections to cluster level switches. The second level switching can span more than 10,000 individual servers.

Programming model for WSC

There is a high variability in performance between the different WSC servers because of:

- varying load on servers
- file may or may not be in a file cache
- distance over network can vary
- hardware anomalies

A WSC will start backup executions on other nodes when tasks have not yet completed and take the result that finishes first. Rely on data replication to help with read performance and availability. A WSC also has to cope with variability in load. Often WSC services are performed with in-house software to reduce costs and optimize for performance.

Storage of WSC

- A WSC uses local disks inside the servers as opposed to network attached storage (NAS).
- The Google file system (GFS) uses local disks and maintains at least three replicas to improve dependability by covering not only disk failures, but also power failures to a rack or a cluster of racks by placing the replicas on different clusters.
- A read is serviced by one of the three replicas, but a write has to go to all three replicas.
- Google uses a relaxed consistency model in that all three replicas have to eventually match, but not all at the same time.

WSC networking

- A WSC uses a hierarchy of networks for interconnection.
- The standard rack holds 48 servers connected by a 48-port Ethernet switch. A rack switch has 2 to 8 uplinks to a higher switch.
- So the bandwidth leaving the rack is $6 (48/8)$ to $24 (48/2)$ times less than the bandwidth within a rack.
- There are array switches that are more expensive to allow higher connectivity.
- There may also be Layer 3 routers to connect the arrays together and to the Internet.
- The goal of the software is to maximize locality of communication relative to the rack.

Performance

Power Utilization Effectiveness (PUE) is widely used metric to estimate the performance of WSCs.

$$\mathbf{PUE} = \frac{\text{Total_utility_power}}{\text{IT_equipment_power}}$$

Bandwidth is an important metric as there may be many simultaneous user requests or metadata generation batch jobs. Latency is also equally important metric as it is seen by users when they make requests. Users will use a search engine less as the response time increases. Also users are more productive in responding to interactive information when the system response time is faster as they are less distracted.

MULTIPROCESSOR NETWORK TOPOLOGIES

Multiprocessor system consists of multiple processing units connected via some interconnection network plus the software needed to make the processing units work together. There are two major factors used to categorize such systems:

- the processing units
- the interconnection network

A number of communication styles exist for multiprocessing networks. These can be broadly classified according to the communication model as shared memory (single address space) versus message passing (multiple address spaces).

Design Issues Of Interconnection Networks

The important issue in the design of multiprocessor systems is how to cope with the problem of an adequate design of the interconnection network in order to achieve the desired performance at low cost. The choice of the interconnection network may affect several characteristics of the system such as node complexity, scalability and cost etc. The following are the issues which should be considered while designing an interconnection network.

- **Dimension and size of network:** It should be decided how many processing element are there in the network and what the dimensionality of the network is i.e. with how many neighbors, each processor is connected.

- **Symmetry of the network:** It is important to consider whether the network is symmetric or not i.e., whether all processors are connected with same number of processing elements or the processing elements of corners or edges have different number of adjacent elements.
- **Message Size:** Message size is dependent on the amount of data that can be transferred in one unit time.
- **Data transfer Time:** The time taken for a message to reach to another processor, Whether this time is a function of link distance between two processors or it depends upon the number of nodes coming in between are chief factors
- **Startup Time:** It is the time of initiation of the process.

Performance parameters

- **Number of nodes (N):** The number of nodes in a multiprocessor network plays a dynamic role by virtue of which the performance of the system is evaluated. Higher number of nodes means higher complexity but higher is the system performance. Therefore, number of processors should be optimal.
- **Node degree(D):** The node degree of the network is defined as the number of edges connected with the nodes. It is the connectivity among different nodes in a network. The connectivity of the nodes determines the complexity of the network. The greater number of links in the network means greater is the complexity. If the edge carries data from the node, it is called out degree and if this carries data into the node then it is called in degree.
- **Diameter (D):**The network diameter is defined as the maximum shortest path between the source and destination node. The path length is measured by the number of links traversed. This virtue is important in determining the distance involved in communication and hence the performance of parallel systems. The low diameter is always better because the diameter puts a lower bound on the complexity of parallel algorithms requiring communication between arbitrary pairs of nodes.
- **Cost (C):** It is defined as the product of the diameter and the degree of the node for a symmetric network.

$$\text{Cost (C)} = \text{Diameter} * \text{Degree} = D * d$$

Greater number of nodes means greater the cost of the network. It is good creation to measure the hardware cost and the performance of the multiprocessor network and gives more insight to design a cost-effective parallel system.

- **Extensibility**

It is virtue which facilitates large sized system out of small ones with minimum changes in the configuration of the nodes. It is the smallest increment by which the system can be expanded in a useful way. A network with large number of links or a large node degree tends to increase the hardware cost. Expandability is an important parameter to evaluate the performance of a multiprocessor system. The feasibility to extend a system while retaining its topological characteristics enables to design large scale parallel systems.

Network Topologies

The multiprocessor networks are classified in two broad categories based on their topological properties. These are given below:

- Cube based network
- Linearly Extensible Network

Cube Based Network

- The cube based architectures are widely used networks in parallel systems. They have good topological properties such as symmetry, scalability and possess a rich interconnection topology. The types of cube based networks are:
 - **Binary hypercube or n-cube:**
 - This is a loosely coupled parallel multiprocessor based on the binary n-cube network.
 - An n-dimensional hypercube contains 2^n nodes and has n edges per node.
 - In hypercube, the number of communication links for each node is a logarithmic function of the total number of nodes.
 - The hypercube organization has low diameter and high bisection width at the expense of the number of edges per node and the length of the longest edge.

- The length of the longest edge in a hypercube network increases as the number of nodes in the network increases.
- The node degree increases exponentially with respect to the dimension, making it difficult to consider the hypercube a scalable architecture.
- The major drawback of the hypercube is the increase in the number of communication links for each node with the increase in the total number of nodes.

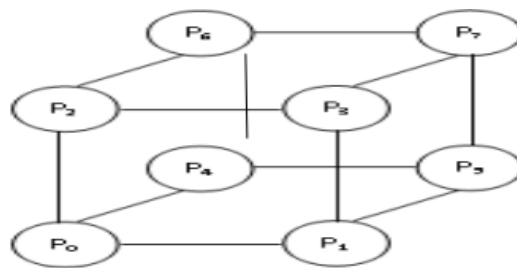


Fig 5.14: Hypercube

• **Cube Connected Cycle (CCC)**

- The CCC architecture is an attractive parallel computation network suitable for VLSI implementation while preserving all the desired features of hypercube.
- The CCC is constructed from the n - dimensional hypercube by replacing each node in hypercube with a ring containing n node.
- Each node in a ring then connects to a distinct node to one of the n dimensions.
- The advantage of the cube- connected cycles is that node's degree is always 3, independent of the value of n . This architecture is modified from hypercube i.e. a 3-cube is modified to form a 3-cube-connected cycles (CCC) restricted the node degree to 3.
- The idea is to replace the corner nodes (vertices) of the 3-cube with a ring of 3-nodes.
- In general one can construct k -cube-connected cycles from a k -cube with $n=2^k$ rings nodes.

- **Folded HyperCube (FHC)**

- The FHC is the variation of the hypercube network and constructed by introducing some extra links to the hypercube.
- Halved diameter, better average distance, shorter delay in communication links, less message traffic density, lower cost make it very promising.
- The hardware overhead is almost $1/n$, n being the dimensionality of the hypercube, which is negligible for large n .
- Optimal routing algorithms are developed and proven to be remarkably more efficient than those of the conventional n -cube.
- A folded hypercube of dimension n is called FHC (n).
- The FHC (n) is a regular network of node connectivity $(n+1)$ and the hypercube of degree 3 is converted to FHC (n) network.
- Extended versions of FHC (n) is called Extended Folded Cube (EFC). The EFC has better properties than the other variations of basic hypercube in terms of parameter.
- It has constant node degree, smaller diameter, and lower cost and also it maintains several numerous desirable characteristics including symmetry, hierarchical, expansive, recursive.

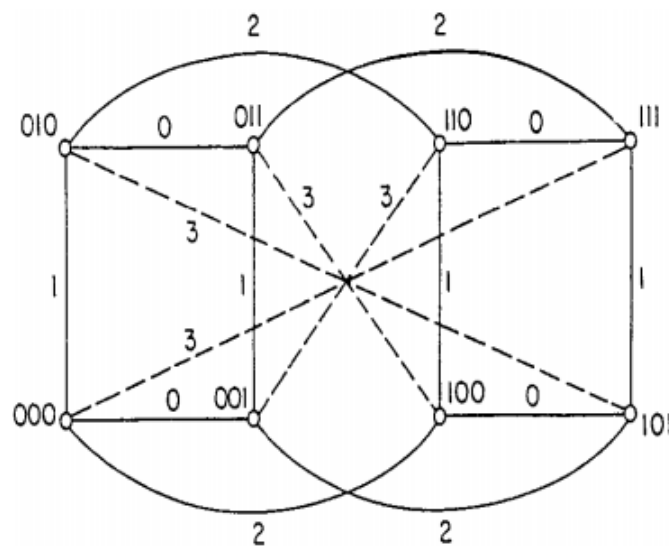


Fig 5.15: Folded Hypercube

• Crossed Cube

- The Crossed Cube (CC) has the same node and link complexity as the hypercube and has most of its desirable properties including regularity, recursive structure, partition ability, strong connectivity and ability to simulate other architectures.
- Its diameter is only half of the diameter of the hypercube.
- Mean distance between vertices is smaller and it can simulate a hypercube through dilation 2 embedding.
- The basic properties of the CC, optimal routing and broadcasting algorithms are developed.
- The CC is derived from a hypercube by changing the way of connection of some hypercube links.
- The diameter of CC is almost half of that of its corresponding hypercube.

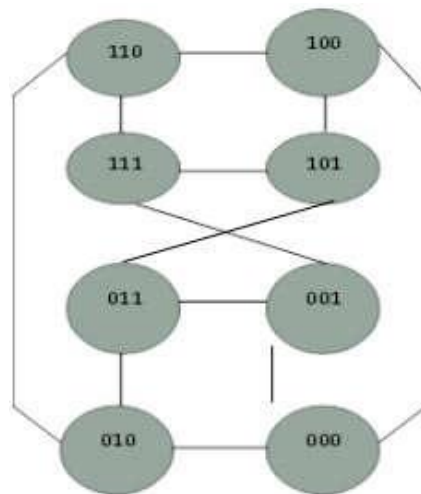


Fig 5.16: Crossed Cube

• Reduced Hypercube (RHC)

- The RH (k, m) is obtained from the n - dimensional hypercube by reducing node edges in hypercube by following rules where $k+2m= n$.
- The lower VLSI complexity of RH's permit the construction of systems with more processing elements than are found in conventional hypercube.
- There are clusters and each cluster is a conventional k - dimensional hypercube.
- Of the higher $n-k=2m$ dimensions, a node has only one direct connection is decided by the leftmost m bits in the k -bits field, i.e., the $(2i + k)$ dimension, where i is the value of the m -bit binary number.

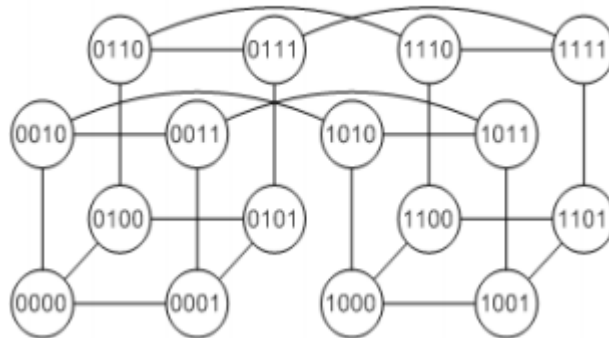


Fig 5.17: Reduced Hypercube

• **Hierarchical Cube Network (HCN)**

- The Hierarchical Cube Network (HCN) is interconnection network for large-scale distributed memory multiprocessors.
- HCN has about three-fourths the diameter of a comparable hypercube, although it uses about half as many links per node—a fact that has positive ramifications on the implementation of HCN-connected systems.
- The HCN (n, n) has $2n$ clusters, where each cluster is an n -cube.
- Each node in the HCN (n, n) has $n+1$ links connected to it. n links are used inside the cluster. The additional links are used to connect nodes among clusters.
- The advantage of HCN is that the number of links required is reduced approximately to half as many links per node and the diameter is reduced to about three-fourth of a corresponding hypercube.

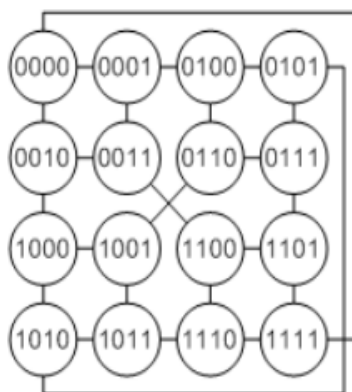


Fig 5.18: Hierarchical Cube Network

- **Dual Cube (DC)**
 - The DC is a new interconnection topology for large-scale distributed memory Multiprocessors that reduces the problem of increasing number of links in the large-scale hypercube network.
 - This preserves most of the topological properties of the hypercube network.
 - The DC shares the desired properties of the hypercube, however increases tremendously the total number of nodes in the system with limited links per node.
 - The key properties of hypercube are also true in the dual-cube: each node can be represented by unique binary number such that two nodes are connected by an edge if and only if the two binary numbers differ in one bit only.
 - However, the size of the dual-cube can be as large as eight thousands with up to eight links per node.
 - A dual-cube uses binary hypercube as basic components. Each such hypercube component is referred to as a **cluster**.
 - Assume that the number of nodes in a cluster is 2^m . In a dual cube, there are two classes with each class consisting of 2^m clusters.
 - The total number of nodes is 2^m or 2^{m+1} . Therefore, the nodes address has $2m+1$ bits
 - The leftmost bit is used to indicate the type of the class (class 0 and class 1).
 - For the class 0, the rightmost m bits are used as the node ID within the cluster.
 - Each node in cluster of class 0 has one and only one extra connection to a node in a cluster of class 1.
- **Meta Cube (MC)**
 - The MC is an interconnection topology for a very large parallel system. Meta cube network has two level cube structures. An MC (k, m) network can connect $2k+m^{2k}$ nodes with $(k+m)$ links per node where k is the dimension of the high-level cubes (classes) and m is the dimension of the low-level cubes (clusters).
 - In this network, the number of nodes is much larger than the hypercube with a small number of links per node.

- An MC network is a symmetric network with short diameter, easy and efficient routing
- similar to that of the hypercube.
- The meta cube has tremendous potential to be used as an interconnection network for very large scale parallel computers since the meta cube can connect hundreds of millions nodes with up to six links per node and it keeps some desired properties of the hypercube that are useful efficient communication among the nodes.

• Folded Dual Cube (FDC)

- The FDC is a new cube based Interconnection topology for parallel systems with reduced diameter, cost and constructed from DC and FHC.
- The FDC is a graph $Fr(V, E)$, where V represents a set of vertices and E represent a set of links.
- The FDC is to be slightly greater than Dualcube but quite less than HC and FHC.
- Diameter of FDC is found to be smaller than that of Dualcube and with the comparison of Dualcube, HC and FHC.
- FDC exhibits quite a good improvement in broadcast time over its parent networks with millions of nodes.
- The cost of the FDC topology is found to be less. The FDC will help to speed up the overall operation of large scale parallel systems.

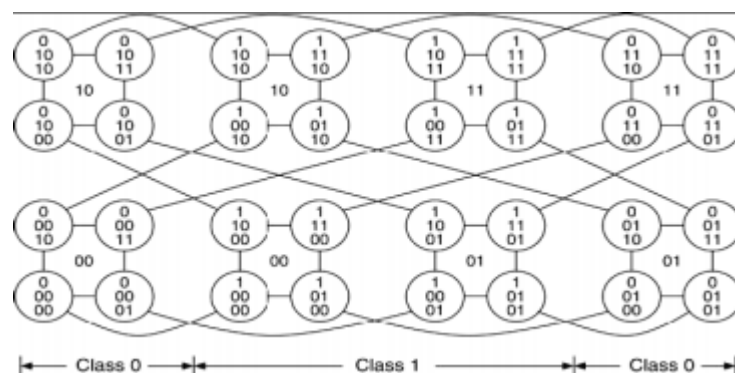


Fig 5.19: Folded Dual Cube

• Folded Metacube (FMC)

- The FMC is an efficient large scale parallel interconnection topology with better features such as reduced diameter, cost, improved broadcast time and constructed from MC.

- The FMC is a graph $G (V, E)$, where V represents a set of vertices and E represent a set of links.
- The FMC is to be slightly greater than metacubebut quite less than HC and FHC.
- Diameter of FMC is found to be smaller than that of Metacube.
- FMC exhibits quite a good improvement in broadcast time over its parent network while connecting millions of nodes.
- The cost of the FMC is found to be less and will help to speed the overall operation of large scale parallel systems.

• **Necklace Hypercube (NH)**

- NH is an array of processors attached to each two adjacent nodes of the hypercube network.
- It is highly scalable architecture while preserving most of the desirable properties of hypercube such as logarithmic diameter, fault tolerance etc.
- It has also some other properties such as hardware scalability and efficient VLSI layout that make it more attractive than an equivalent hypercube network.
- The Necklace-Hypercube is an undirected graph which has a necklace of processors to each edge of hypercube.
- The necklace length may be fixed or variable for different edge necklaces.

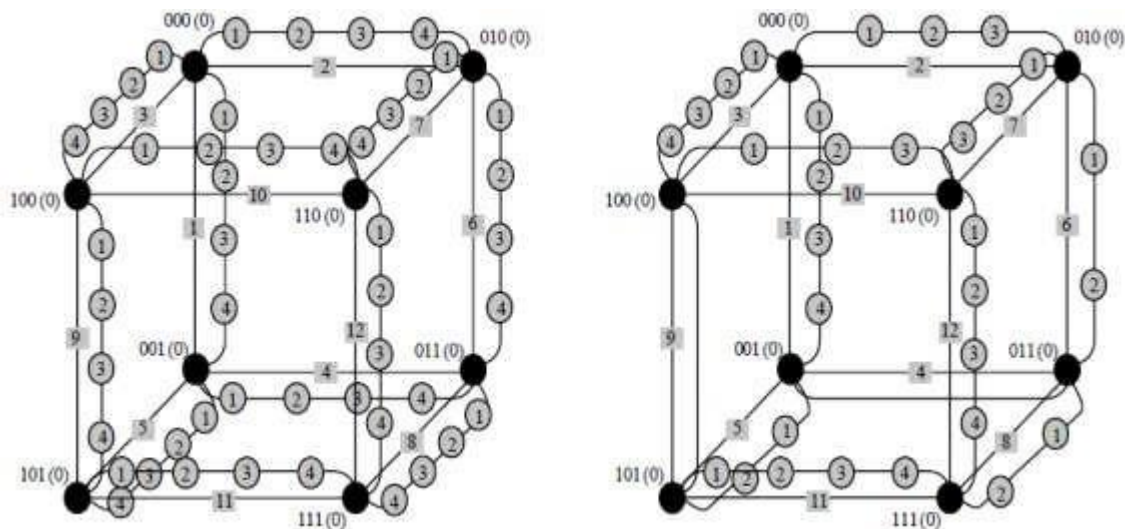


Fig 5.20: Necklace Hypercube

Linearly Extensible Network

The Linearly Extensible Networks is another class of multiprocessor architectures which reduces some of the drawbacks of HC architectures. The complexity of these networks is lesser as they do not have exponential expansion. Besides the scalability, other parameters to evaluate the performance of such networks are degree, number of nodes, diameter, bisection width and fault tolerance. Selection of a better interconnection network may have several applications with lesser complexities and improved power-efficiency.

- **Linear Array (LA)**

- It is one dimensional network having the simplest topology with n -nodes having $n-1$ communication links.
- The internal nodes have degree 2 and the termination nodes have degree 1.
- The diameter is $n-1$, which is long for large n and the bisection width is 1.
- It is asymmetric network. Linear array are the simplest connection topology.
- As the diameter increases linearly with respect to n , it should not be used for large n . For every small n , it is rather economical to implement a linear array.

- **Binary Tree (BT)**

- A binary tree is either empty or consists of node called the root together with two binary trees called left subtree and the right subtree.
- When h is equal to height of a binary tree then maximum leaves are equal to 2^h and maximum nodes are $2^{h+1}-1$.
- In a binary tree network there is only one path between any two nodes.
- The binary tree is scalable architecture with a constant node degree and constant bisection width. In general, an n -level, complexity balanced binary tree should have $N=2n-1$ nodes.
- The maximum node degree is 3 and the diameter is $2(n-1)$. But has a poor bisection width of 1.

• Ring (R)

- This is a simple linear array where the end nodes are connected. It is equivalent to mesh with wrap around connections.
- The data transfer in a ring is normal one direction. A ring is obtained by connecting the two terminal nodes of a linear array with one extra link.
- A ring network can be uni-or bidirectional and it is symmetric with a constant.
- It has a constant node degree of $d=2$, the diameter is $N/2$ for a bidirectional ring and N for unidirectional ring.
- A ring network has a constant width 2.

• Linearly Extensible Tree (LET)

- The Linearly Extensible Tree (LET) architecture exhibits better connectivity, lesser number of nodes over cube based networks.
- The LET network has low diameter, hence reduce the average pathlength traveled by all message and contains a constant degree per node.
- The LET network grows linearly in a binary tree like shape.
- In a binary tree the number of nodes at level n is 2^n whereas in LET network the number is $(n+1)$.

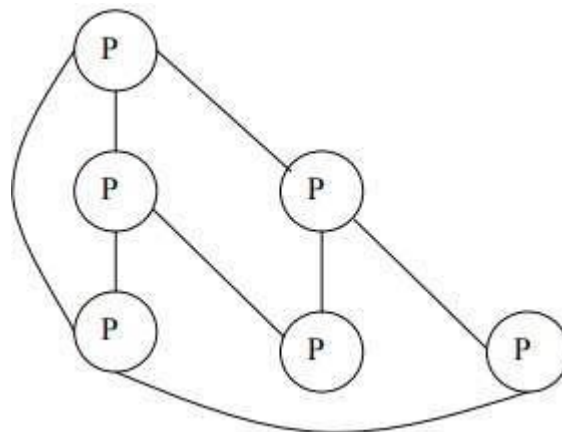
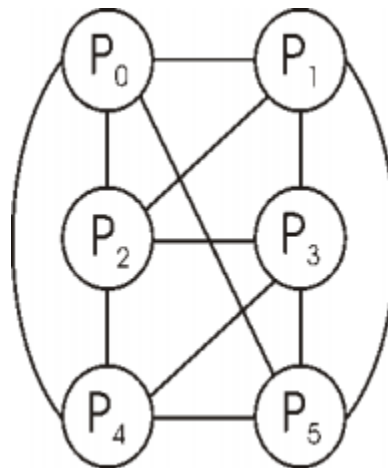


Fig 5.21: Linearly Extensible Tree

Linearly Extensible Cube (LEC)

- The LEC network grows linearly and possesses some of the desirable topological properties such as small diameter, high connecting constant node degree with high scalability.
- It has constant expansion of only two processors at each level of the extension while preserving all the desirable topological properties.
- The LEC network can maintain a constant node degree regardless of the increase in size in a network.
- The number of nodes in LEC network is $2 * n$ for $n > 0$ whereas the number of nodes in the hypercube is 2^n . The diameter of network is N . It has a constant node degree 4.

**Fig 5.22:Linearly Extensible Cube**



MADHA
Expertise | Empathy | Excellence
ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COMMON FOR: DEPARTMENT OF INFORMATION TECHNOLOGY

CS8492 – DATABASE MANAGEMENT SYSTEMS

YEAR / SEM : II / III

R – 2017

LECTURE NOTES

Unit – III Transactions

Transaction Concepts - ACID Properties - Schedules - Serializability - Concurrency Control - Need for Concurrency- Locking Protocols- Two Phase Locking- Deadlock- Transaction Recovery - Save Points - Isolation Levels - SQL Facilities for Concurrency and Recovery.

Introduction

- ❑ **Single-User System:** Only one user can use the system at a time.
- ❑ **Multiuser System:** Many users can access the system concurrently.

Concurrency

- ❑ **Interleaved Processing:** Concurrent execution of processes is interleaved in a single CPU.
- ❑ **Parallel Processing:** Processes are concurrently executed in multiple CPUs.

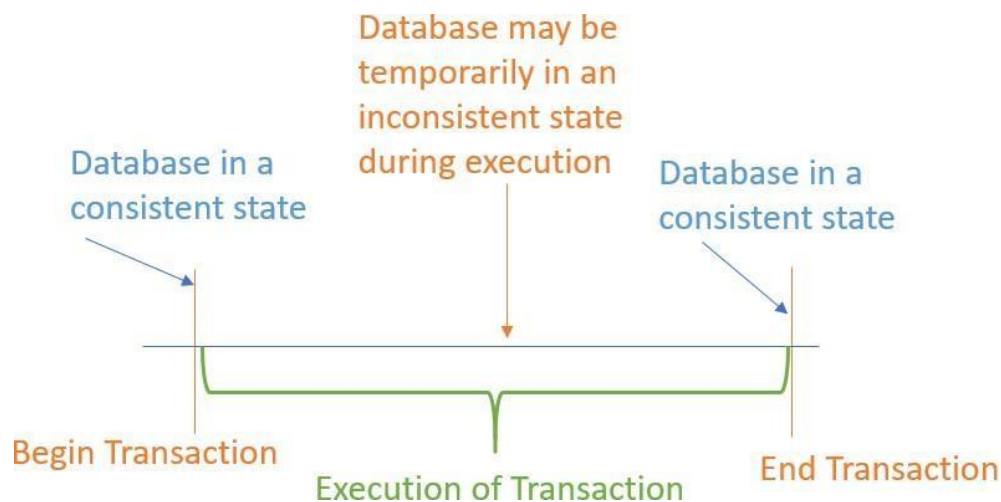
Transaction Concepts

Write short notes on transaction concepts. (Nov/Dec 2014)

- ❑ A **transaction** can be defined as a group of tasks. It is a logical unit of work on the database processing that includes one or more access operations (read-retrieval, write-insert or update, delete).

(Or)

- ❑ Collections of operations that form a single logical unit of work are called **transactions**.
- A database system must ensure proper execution of transactions despite failures—either the entire transaction executes, or none of it does.
- ❑ Usually, a transaction is initiated by a user program written in a high-level data-manipulation language (typically SQL), or programming language (for example, C++, or Java), with embedded database accesses in JDBC or ODBC.
- ❑ A transaction is delimited by statements (or function calls) of the form **begin transaction** and **end transaction**.
- ❑ The transaction consists of all operations executed between the **begin transaction** and **end transaction**.



A Simple Transaction Model / Simple Model of a Database (for purposes of transactions):

- A database - collection of named data items
- Granularity of data - a field, a record or a whole disk block (Concepts are independent of granularity)
- Basic operations are read and write.
 - read_item(X): Reads a database item named X into a program variable. To simplify our notation, we assume that *the program variable is also named X*.
 - write_item(X): Writes the value of program variable X into the database item named X.

Read Operation:

- Basic unit of data transfer from the disk to the computer main memory is one block.
- In general, a data item (what is read or written) will be the field of some record in the database, although it may be a larger unit such as a record or even a whole block.
- read_item(X) command includes the following steps:
 - ✓ Find the address of the disk block that contains item X.
 - ✓ Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
 - ✓ Copy item X from the buffer to the program variable named X.

Write Operation:

- write_item(X) command includes the following steps:
 - ✓ Find the address of the disk block that contains item X.
 - ✓ Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
 - ✓ Copy item X from the program variable named X into its correct location in the buffer.
 - ✓ Store the updated block from the buffer back to disk (either immediately or at some later point in time).

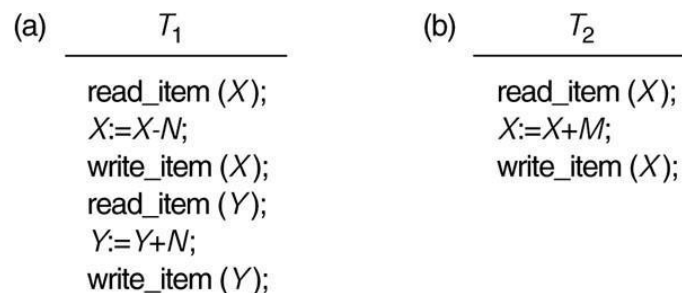


Figure: Two sample transactions. (a) Transaction T_1 (b) Transaction T_2

- Let T_i be a transaction that transfers \$50 from account A to account B. This transaction can be defined as:
 T_i : read(A);
 $A := A - 50$;
 write(A);
 read(B);

$B:=B+50;$

write(B).

Storage Structure

- To understand how to ensure the atomicity and durability properties of a transaction, we must gain a better understanding of how the various data items in the database may be stored and accessed.

Volatile Storage

- Information residing in volatile storage does not usually survive system crashes.
- Examples of such storage are main memory and cache memory.

Nonvolatile Storage

- Information residing in nonvolatile storage survives system crashes.
- Examples of nonvolatile storage include secondary storage devices such as magnetic disk and flash storage, used for online storage, and tertiary storage devices such as optical media, and magnetic tapes, used for archival storage.

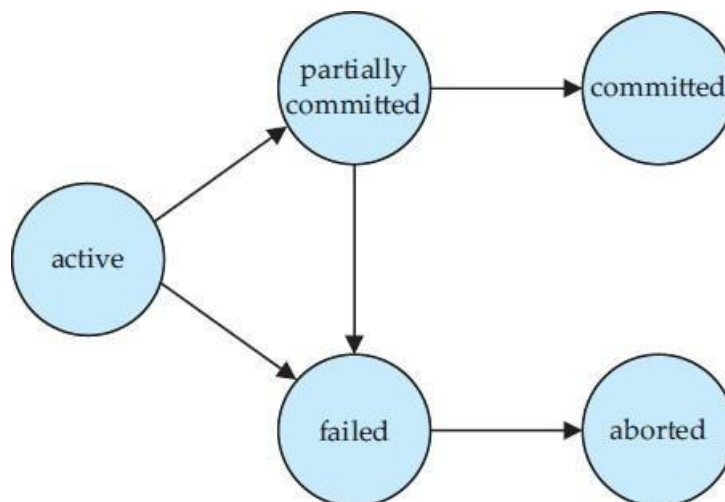
Stable Storage

- Information residing in stable storage is *never* lost (*never* should be taken with a grain of salt, since theoretically *never* cannot be guaranteed).

State Diagram of a Transaction

Write short notes on states of a transaction.

- A transaction in a database can be in one of the following states:
 - ✓ Active
 - ✓ Partially Committed
 - ✓ Failed
 - ✓ Aborted
 - ✓ Committed



(Or)

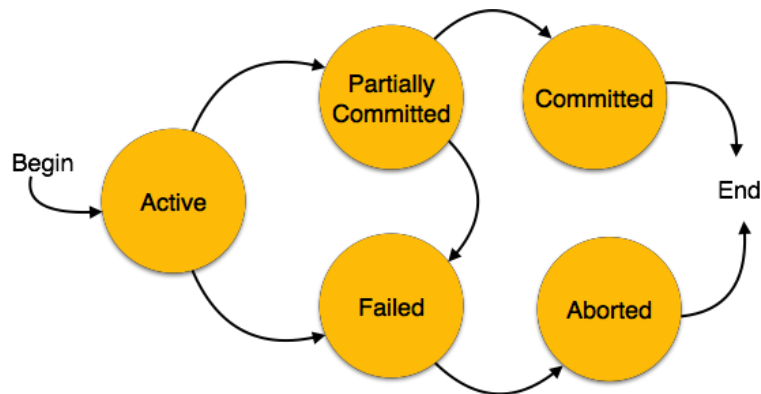


Figure: State Diagram of a Transaction

Active State

- The active state is the first state of every transaction.
- In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

Partially Committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

Committed

- A transaction is said to be in a committed state if it executes all its operations successfully.
- In this state, all the effects are now permanently saved on the database system.

Failed State

- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state.
- If not then it will abort or rollback the transaction to bring the database into a consistent state.
- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
 - ✓ Re-start the Transaction
 - ✓ Kill the Transaction

ACID Properties

Explain with an example the properties that must be satisfied by a transaction. (April/May 2018)

- The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Properties of Transaction

1. Atomicity
2. Consistency
3. Isolation
4. Durability

Atomicity

- It states that all operations of the transaction take place at once; if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either runs to completion or is not executed at all.
- Atomicity involves the following two operations:
 - ✓ **Abort:** If a transaction aborts then all the changes made are not visible.
 - ✓ **Commit:** If a transaction commits then all the changes made are visible.

Example:

- Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2
Read(A)	Read(B)
A:= A-100	Y:= Y+100
Write(A)	Write(B)

- After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.
- If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B.
- This shows the inconsistent database state.
- In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

- Forexample:Thetotalamountmustbemaintainedbeforeorafterthetransaction.

1. Total before T occurs = 600 + 300 = 900

2. Total after T occurs = 500 + 400 = 900

- Therefore,thedatabaseisconsistent.InthecasewhenT1iscompletedbutT2fails,then inconsistency will occur.

Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforces the isolation property.

Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure.
- When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

Schedules

Explain in detail about schedules with an example.

- A series of operations from one transaction to another transaction is known as a schedule.
- It is used to preserve the order of the operation in each of the individual transactions.
- If two transactions are executed at the same time, the result of one transaction may affect the output of the other.

Example

Initial Product Quantity is 10

Transaction 1: Update Product Quantity to 50

Transaction 2: Read Product Quantity

- If Transaction 2 is executed before Transaction 1, outdated information about the product quantity will be read. Hence, schedules are required.

Equivalence Schedules

- Parallel execution in a database is inevitable.
- But, Parallel execution is permitted when there is an equivalence relation among the simultaneously executing transactions.
- This equivalence is of 3 Types.

Result Equivalence

- If two schedules display the same result after execution, it is called a result equivalent schedule.

- They may offer the same result for some value and different results for another set of values.
- For example, one transaction updates the product quantity, while other updates customer details.

View Equivalence

- View Equivalence occurs when the transaction in both the schedule performs a similar action.
- For example, one transaction inserts product details in the product table, while another transaction inserts product details in the archive table.
- The transaction is the same, but the tables are different.

Conflict Equivalence

- In this case, two transactions update/view the same set of data.
- There is a conflict among transactions as the order of execution will affect the output.

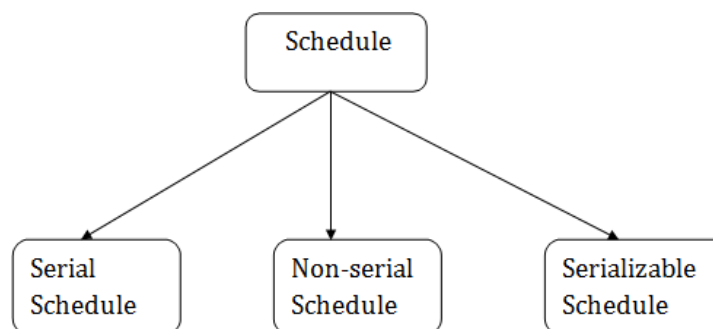
Two schedules would be conflicting if they have the following properties –

- Both belong to separate transactions.
- Both access the same data item.
- At least one of them is "write" operation.

Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if –

- Both the schedules contain the same set of Transactions.
- The order of conflicting pairs of operation is maintained in both the schedules.

Types of Schedule

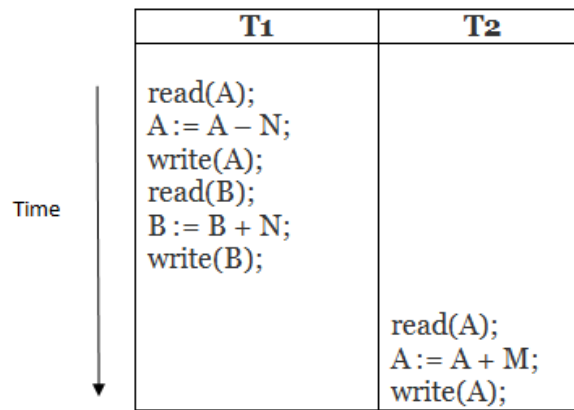


Serial Schedule

- The serial schedule is a type of schedule where one transaction is executed completely before starting another transaction.
- In the serial schedule, when the first transaction completes its cycle, then the next transaction is executed.
- For example: Suppose there are two transactions T1 and T2 which have some operations.
- If there is no interleaving of operations, then there are the following two possible outcomes:
 - ✓ Execute all the operations of T1 which was followed by all the operations of T2.
 - ✓ Execute all the operations of T2 which was followed by all the operations of T1.
- In the given (a) figure, Schedule A shows the serial schedule where T1 followed by T2.

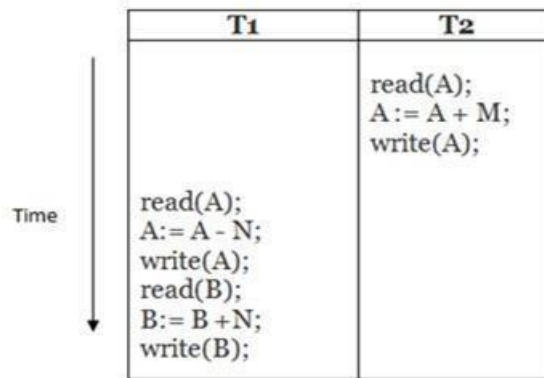
□ In the given (b) figure, Schedule B shows the serial schedule where T2 followed by T1.

(a)



Schedule A

(b)



Schedule B

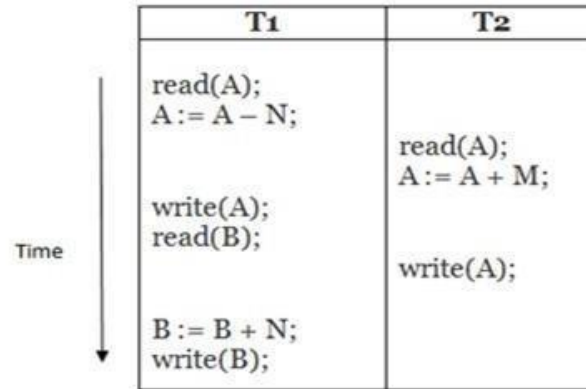
Here,

□ Schedule A and Schedule B are serial schedule.

Non-serial Schedule

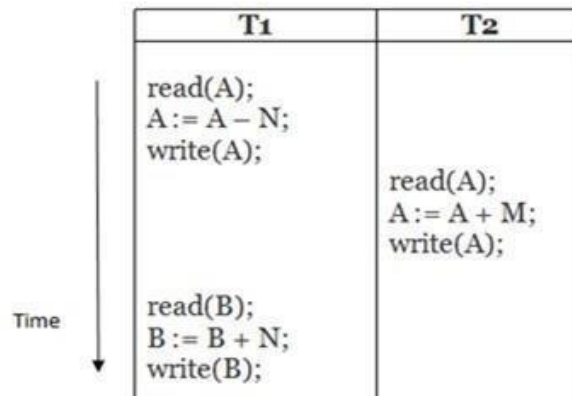
- If interleaving of operations is allowed, then there will be non-serial schedule.
- It contains many possible orders in which the system can execute the individual operations of the transactions.
- In the given figure (c) and (d), Schedule C and Schedule D are non-serial schedules.
- It has interleaving of operations.

(c)



Schedule C

(d)



Schedule D

Here,

- Schedule C and Schedule D are Non-serial schedule.

Serializable Schedule

- The serializability of schedules is used to find non-serial schedules that allow the transaction to execute concurrently without interfering with one another.
- It identifies which schedules are correct when executions of the transaction have interleaving of their operations.
- A non-serial schedule will be serializable if its result is equal to the result of its transactions executed serially.

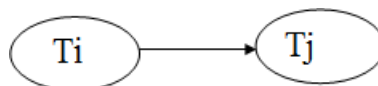
Serializability

Explain Serializability in detail. (Or) Discuss View Serializability and Conflict Serializability. (Nov/Dec 2015, April/May 2018)

- **Serializability** is a concurrency scheme where the concurrent transaction is equivalent to one that executes the transactions serially.
- A schedule is a list of transactions.
- The objective of a **concurrency control** protocol is to schedule transactions in such a way as to avoid any interference between them.
- **Schedule** is a sequence of the operations by a set of concurrent transactions that preserves the order of the operations in each of the individual transactions.
- **Serial schedule** is a schedule where the operations of each transaction are executed consecutively without any interleaved operations from other transactions.
- In a serial schedule, the transactions are performed in serial order, i.e., if T1 and T2 are transactions, serial order would be T1 followed by T2 or T2 followed by T1.
- **Non serial schedule** is a schedule where the operations from a set of concurrent transactions are interleaved.
- In non-serial schedule, if the schedule is not proper, then the problems can arise like multiple update, uncommitted dependency and incorrect analysis.
- The **objective of serializability** is to find non serial schedules that allow transactions to execute concurrently without interfering with one another, and thereby produce a database state that could be produced by a serial execution.

Testing of Serializability

- Serialization Graph is used to test the Serializability of a schedule.
- Assume a schedule S. For S, we construct a graph known as precedence graph.
- This graph has a pair $G = (V, E)$, where V consists a set of vertices, and E consists a set of edges. The set of vertices is used to contain all the transactions participating in the schedule.
- This set of edges is used to contain all edges $T_i \rightarrow T_j$ for which one of the three conditions holds:
 1. Create a node $T_i \rightarrow T_j$ if T_i executes write (Q) before T_j executes read (Q).
 2. Create a node $T_i \rightarrow T_j$ if T_i executes read (Q) before T_j executes write (Q).
 3. Create a node $T_i \rightarrow T_j$ if T_i executes write (Q) before T_j executes write (Q).

Precedence Graph for Schedule S:

- If a precedence graph contains a single edge $T_i \rightarrow T_j$, then all the instructions of T_i are executed before the first instruction of T_j is executed.

- If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

For Example:

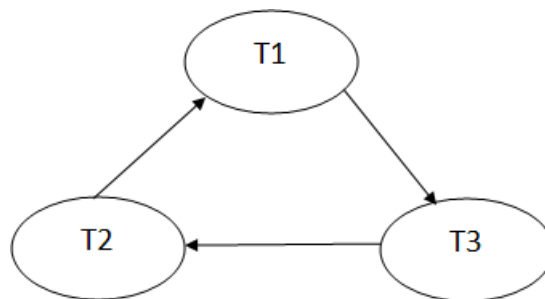
	T1	T2	T3
Time ↓	Read(A)	Read(B)	
	A:=f1(A)		Read(C)
		B:= f2(B) Write(B)	C:= f3(C) Write(C)
	Write(A)	Read(A) A:= f4(A)	
	Read(C)	Write(A)	
	C:= f5(C) Write(C)		B:= f6(B) Write(B)

Schedule S1

Explanation:

- **Read(A):** In T1, no subsequent writes to A, so no new edges
- **Read(B):** In T2, no subsequent writes to B, so no new edges
- **Read(C):** In T3, no subsequent writes to C, so no new edges
- **Write(B):** B is subsequently read by T3, so added edge T2 → T3
- **Write(C):** C is subsequently read by T1, so added edge T3 → T1
- **Write(A):** A is subsequently read by T2, so added edge T1 → T2
- **Write(A):** In T2, no subsequent reads to A, so no new edges
- **Write(C):** In T1, no subsequent reads to C, so no new edges
- **Write(B):** In T3, no subsequent reads to B, so no new edges

Precedence Graph for Schedule S1:



- The precedence graph for schedule S1 contains a cycle that's why Schedule S1 is non-serializable.

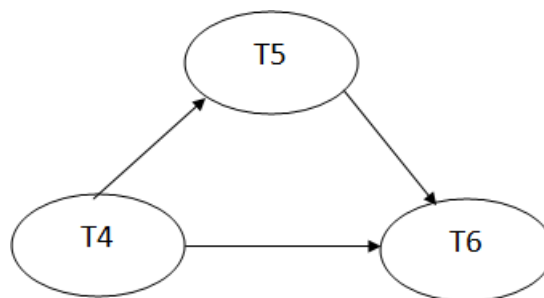
	T4	T5	T6
Time ↓	Read(A)		
	A:= f1(A)		
	Read(C)		
	Write(A)		
	A:= f2(C)		
		Read(B)	
	Write(C)	Read(A)	
		B:= f3(B)	Read(C)
		Write(B)	C:= f4(C)
		A:=f5(A)	Read(B)
	Write(A)	Write(C)	
		B:= f6(B)	
		Write(B)	

Schedule S2

Explanation:

- Read(A):** InT4,nosubsequentwritestoA,sononewedges
- Read(C):** InT4,nosubsequentwritestoC,sononewedges
- **Write(A):** A is subsequently read by T5, so add edge T4 → T5
- Read(B):** In T5,no subsequent writes to B, so no new edges
- **Write(C):** Cissubsequently readbyT6,soaddedgeT4→T6
- **Write(B):** A is subsequentlyreadbyT6,soaddedgeT5→T6
- Write(C):** InT6,nosubsequentreadstoC,sononewedges
- Write(A):** InT5,nosubsequentreadstoA,sononewedges
- Write(B):** In T6, no subsequent reads to B, so no new edges

Precedence graph for schedule S2:



- The precedence graph for schedule S2 contains no cycle that's why ScheduleS2 is serializable.

Types of Serializability

1. Conflict Serializability

- ┆ Conflict serializability defines two instructions of two different transactions accessing the same data item to perform a read/write operation.
- ┆ Itdealswithdetectingtheinstructionsthatareconflictinginanywayandspecifyingthe order in which the instructions should execute in case there is any conflict.

- ┆ A conflict serializability arises when one of the instruction is a write operation. The following rules are important in Conflict Serializability,
- ┆ If two transactions are both read operation, then they are not in conflict.
- ┆ If one transaction wants to perform a read operation and other transaction wants to perform a write operation, then they are in conflict and cannot be swapped.
- ┆ If both the transactions are for write operation, then they are in conflict, but can be allowed to take place in any order, because the transactions do not read the value updated by each other.

2. View Serializability

- ┆ View serializability is another type of serializability.
- ┆ It can be derived by creating another schedule out of an existing schedule and involves the same set of transactions.

Example:

- ┆ Let us assume two transactions T1 and T2 that are being serialized to create two different schedules S1 and S2, where T1 and T2 want to access the same data item.
- ┆ Now there can be three scenarios,
 - ✓ If in S1, T1 reads the initial value of data item, then in S2, T1 should read the initial value of that same data item.
 - ✓ If in S2, T1 writes a value in the data item which is read by T2, and then in S2, T1 should write the value in the data item before T2 reads it.
- ┆ If in S1, T1 performs the final write operation on that data item, then in S2, T1 should perform the final write operation on that data item.
- ┆ If a concurrent schedule is view equivalent to a serial schedule of same transaction then it is said to be View serializable.

Conflict Serializable Schedule

- ┆ A schedule is called conflict serializability if after swapping of non-conflicting operations, it can transform into a serial schedule.
- ┆ The schedule will be a conflict serializable if it is conflict equivalent to a serial schedule.

Conflicting Operations

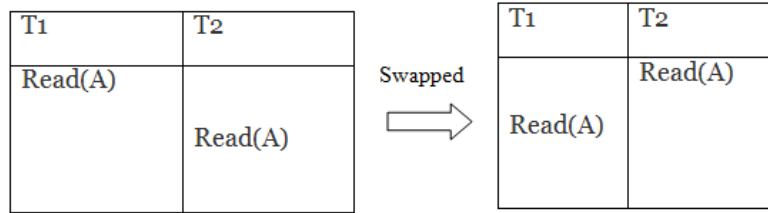
The two operations become conflicting if all conditions satisfy:

1. Both belong to separate transactions.
2. They have the same data item.
3. They contain at least one write operation.

Example:

- ┆ Swapping is possible only if S1 and S2 are logically equal.

1. T1: Read(A) T2: Read(A)

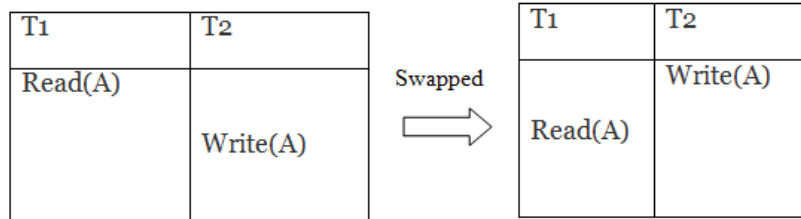


Schedule S1

Schedule S2

┆ Here, $S1 = S2$. That means it is non-conflict.

2. T1: Read(A) T2: Write(A)



Schedule S1

Schedule S2

• Here, $S1 \neq S2$. That means it is conflict.

Conflict Equivalent

- ┆ In the conflict equivalent, one can be transformed to another by swapping non-conflicting operations.
- ┆ In the given example, S2 is conflict equivalent to S1 (S1 can be converted to S2 by swapping non-conflicting operations).

Two schedules are said to be conflict equivalent if and only if:

1. They contain the same set of the transaction.
2. If each pair of conflict operations are ordered in the same way.

Example:

Non-serial schedule

T1	T2
Read(A)	
Write(A)	
	Read(A)
	Write(A)
Read(B)	
Write(B)	
	Read(B)
	Write(B)

Schedule S1

Serial Schedule

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

Schedule S2

- ┆ Schedule S2 is a serial schedule because, in this, all operations of T1 are performed before starting any operation of T2.
- ┆ Schedule S1 can be transformed into a serial schedule by swapping non-conflicting operations of S1.
- ┆ After swapping of non-conflict operations, the schedule S1 becomes:

T1	T2
Read(A)	
Write(A)	
Read(B)	
Write(B)	
	Read(A)
	Write(A)
	Read(B)
	Write(B)

- ┆ Since, S1 is conflict serializable.

View Serializability

- ┆ A schedule will be view serializable if it is view equivalent to a serial schedule.
- ┆ If a schedule is conflict serializable, then it will be view serializable.
- ┆ The view serializable which does not conflict serializable contains blind writes.

View Equivalent

- ┆ Two schedules S1 and S2 are said to be view equivalent if they satisfy the following conditions:

1. Initial Read

- ┆ An initial read of both schedules must be the same.
- ┆ Suppose two schedule S1 and S2.
- ┆ In schedule S1, if a transaction T1 is reading the data item A, then in S2, transaction T1 should also read A.

T1	T2
Read(A)	
	Write(A)

Schedule S1

T1	T2
Read(A)	Write(A)

Schedule S2

- ┆ Above two schedules are view equivalent because Initial read operation in S1 is done by T1 and in S2 it is also done by T1.

2. Updated Read

- ┆ In schedule S1, if Ti is reading A which is updated by Tj then in S2 also, Ti should read A which is updated by Tj.

T1	T2	T3
Write(A)	Write(A)	
		Read(A)

Schedule S1

T1	T2	T3
Write(A)	Write(A)	
		Read(A)

Schedule S2

- Abovetwoschedulesarenotviewequalbecause,inS1,T3isreadingAupdatedbyT2and in S2, T3 is reading A updated by T1.

3. Final Write

- A final write must be the same between both the schedules. In schedule S1, if a transaction T1 updates A at last then in S2, finalwritesshouldalsobedoneby T1.

T1	T2	T3
Write(A)	Read(A)	
		Write(A)

Schedule S1

T1	T2	T3
Write(A)	Read(A)	
		Write(A)

Schedule S2

- AbovetwoschedulesisviewequalbecauseFinalwriteoperationinS1isdonebyT3andin S2, the final write operation is also done by T3.

Example:

T1	T2	T3
Read(A)	Write(A)	
Write(A)		Write(A)

Schedule S

- With3transactions,thetotalnumberofpossibleschedule $1. = 3! = 6$
- 2. S1=<T1T2T3>
- 3. S2=<T1T3T2>
- 4. S3=<T2T3T1>
- 5. S4=<T2T1T3>
- 6. S5=<T3T1T2>
- 7. S6=<T3T2T1>

Taking first schedule S1:

T1	T2	T3
Read(A) Write(A)	Write(A)	
		Write(A)

Schedule S1**Step 1: Final Updation on Data Items**

- ┆ In both schedules S and S1, there is no read except the initial read that's why we don't need to check that condition.

Step 2: Initial Read

- ┆ The initial read operation in S is done by T1 and in S1, it is also done by T1.

Step 3: Final Write

- ┆ The final write operation in S is done by T3 and in S1, it is also done by T3. So, S and S1 are view Equivalent.
- ┆ The first schedule S1 satisfies all three conditions, so we don't need to check another schedule.
- ┆ Hence, view equivalent serial schedule is:

T1 → T2 → T3

Recoverability of Schedule

- ┆ Sometimes a transaction may not execute completely due to a software issue, system crash or hardware failure.
- ┆ In that case, the failed transaction has to be rollback.
- ┆ But some other transaction may also have used value produced by the failed transaction.
- ┆ So we also have to rollback those transactions.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A = A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		
Failure Point				
Commit;				

- ┆ The above table 1 shows a schedule which has two transactions.
- ┆ T1 reads and writes the value of A and that value is read and written by T2.
- ┆ T2 commits but later on, T1 fails. Due to the failure, we have to rollback T1.
- ┆ T2 should also be rollback because it reads the value written by T1, but T2 can't be rollback because it already committed.
- ┆ So this type of schedule is known as irrecoverable schedule.

Irrecoverable Schedule

- ┆ The schedule will be irrecoverable if Tj reads the updated value of Ti and Tj committed before Ti commit.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
		Read(A);	A = 6000	A = 6000
		A = A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
Failure Point				
Commit;				
		Commit;		

- ┆ The above table 2 shows a schedule with two transactions.
- ┆ Transaction T1 reads and writes A, and that value is read and written by transaction T2.
- ┆ But later on, T1 fails. Due to this, we have to rollback T1.
- ┆ T2 should be rollback because T2 has read the value written by T1.
- ┆ As it has not committed before T1 commits so we can rollback transaction T2 as well.
- ┆ So it is recoverable with cascade rollback.

Recoverable with Cascading Rollback

- ┆ The schedule will be recoverable with cascading rollback if Tj reads the updated value of Ti. Commit of Tj is delayed till commit of Ti.

T1	T1's buffer space	T2	T2's buffer space	Database
				A = 6500
Read(A);	A = 6500			A = 6500
A = A - 500;	A = 6000			A = 6500
Write(A);	A = 6000			A = 6000
Commit;		Read(A);	A = 6000	A = 6000
		A = A + 1000;	A = 7000	A = 6000
		Write(A);	A = 7000	A = 7000
		Commit;		

- ┆ The above Table 3 shows a schedule with two transactions.
- ┆ Transaction T1 reads and write A and commits, and that value is read and written by T2.
- ┆ So this is a cascade less recoverable schedule.

Concurrency Control

What is Concurrency? Explain it in terms of locking mechanism and two phase commit protocols. (Nov/Dec 2014) (Or) What is Concurrency Control? How is it implemented in DBMS? Illustrate with a suitable example. (Nov/Dec 2015) (Or) State and explain the lock based concurrency control with suitable example. (Nov/Dec 2017)

- ┆ Concurrency control manages the transactions simultaneously without letting them interfere with each another.
- ┆ The **main objective of concurrency** is to **allow multiple users to perform** different operations at the same time.

(Or)

- ┆ A mechanism which ensures that simultaneous execution of more than one transaction does not lead to any database inconsistencies is called **concurrency control mechanism**.
- ┆ Using more than one transaction concurrently improves the performance of system.

- ┆ If we are not able to perform the operations concurrently, then there can be serious problems such as loss of data integrity and consistency.
- ┆ Concurrency control increases the throughput because of handling multiple transactions simultaneously.
- ┆ It reduces waiting time of transaction.

Purpose of Concurrency Control

- [The fundamental goal of database concurrency control is to ensure that concurrent execution of transactions does not result in a loss of database consistency.
- [The concept of serializability can be used to achieve this goal, since all serializable schedules preserve consistency of the database.
- [To enforce Isolation (through mutual exclusion) among conflicting transactions.
- [To preserve database consistency through consistency preserving execution of transactions.
- [To resolve read-write and write-write conflicts.

Example:

- [In concurrent execution environment if T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.
- [Different types of protocols/schemes used to control concurrent execution of transactions are:
 - ✓ Lock based Protocols
 - ✓ Timestamp based Protocols
 - ✓ Graph based Protocols

Need for Concurrency

Explain why Concurrency Control is needed?

- [The purposes of concurrency control are,
 - ✓ To ensure isolation
 - ✓ To resolve read-write or write-write conflicts
 - ✓ To preserve consistency of database

The Lost Update Problem

- [This occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

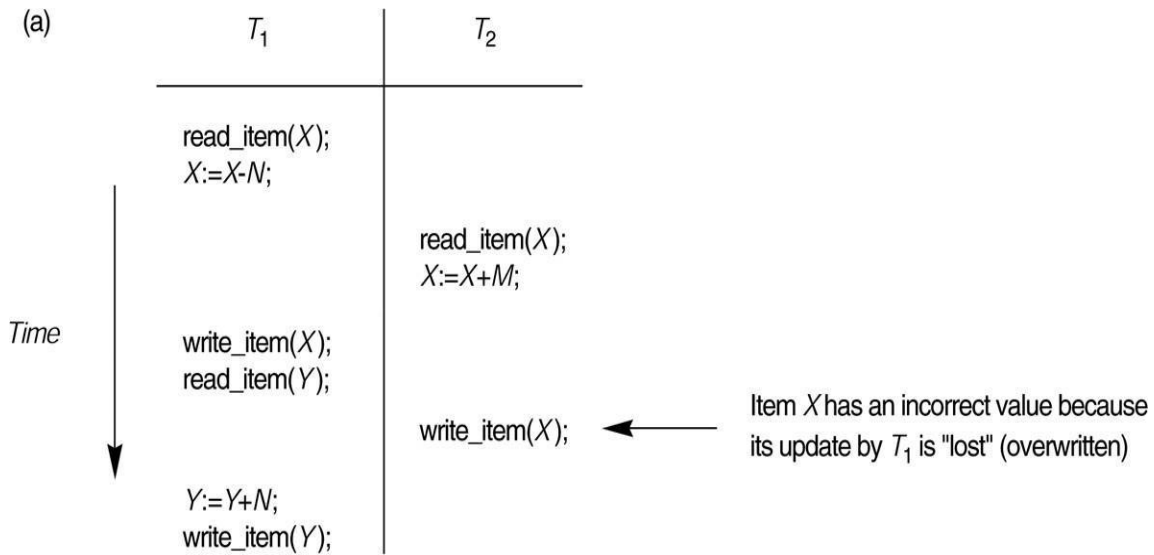
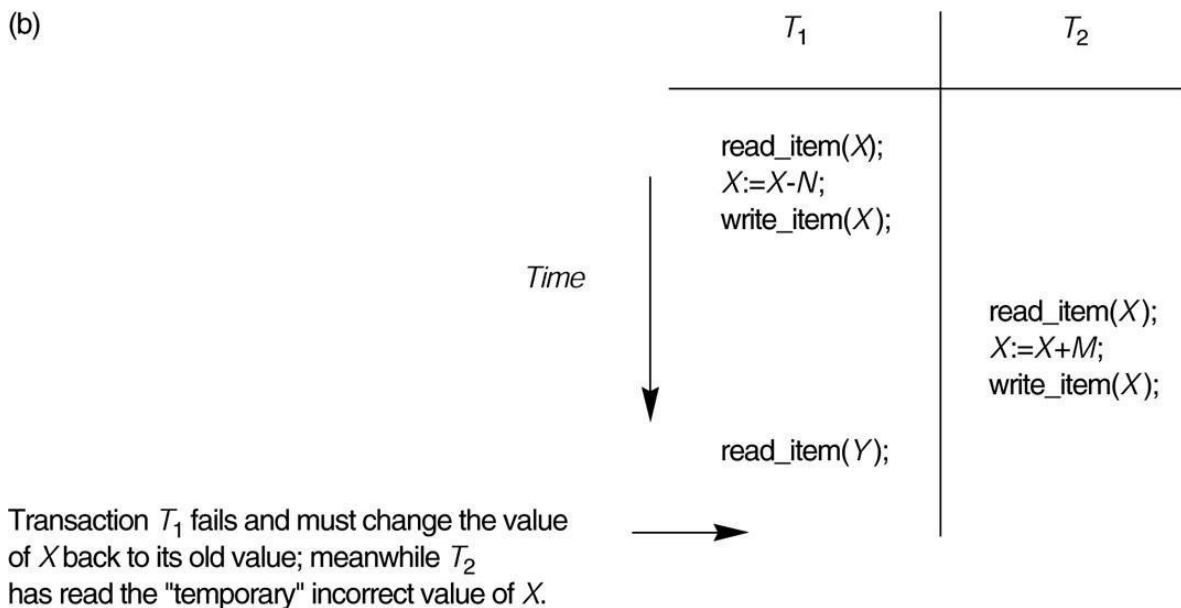


Figure: (a) The Lost Update Problem

The Temporary Update (or Dirty Read) Problem

- This occurs when one transaction updates a database item and then the transaction fails for some reason.
- The updated item is accessed by another transaction before it is changed back to its original value.

Figure: (b) The Temporary Update Problem



The Incorrect Summary Problem

- If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.

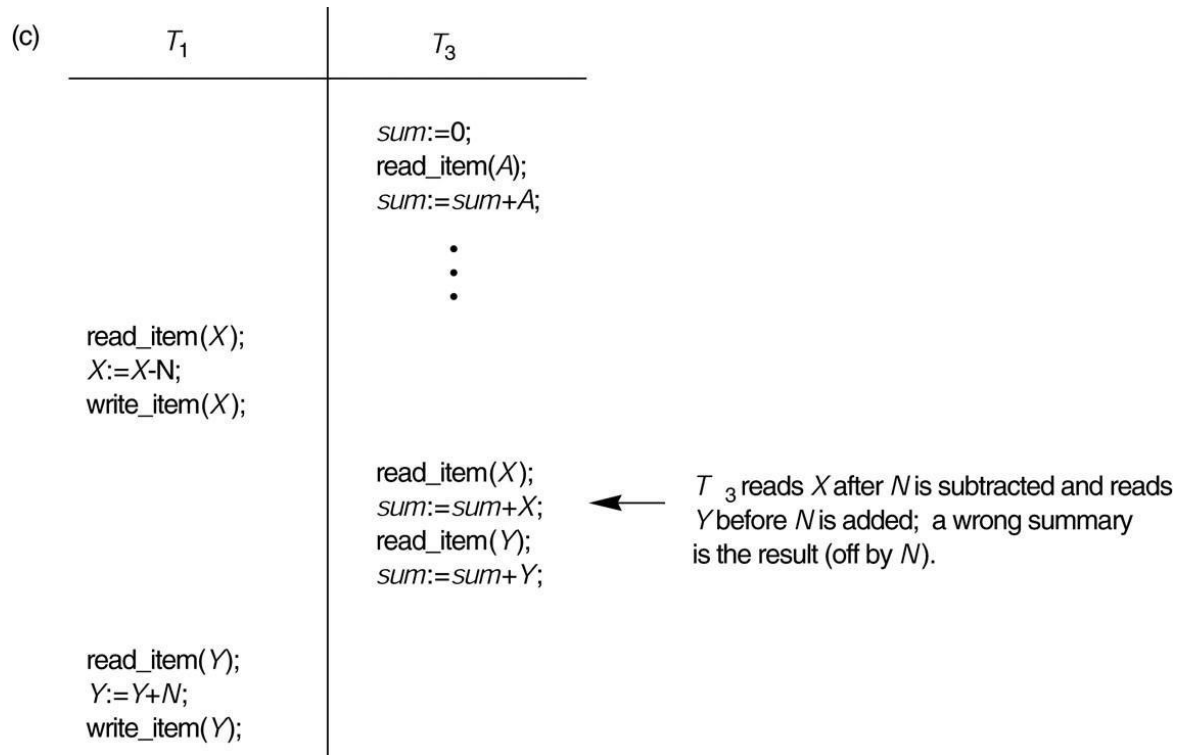


Figure: (c) The Incorrect Summary Problem

Locking Protocols

Explain in detail about various locking protocols.

Locking

- **Locking** is a procedure used to control concurrent access to data when one transaction is accessing the database, a lock may deny access to other transactions to prevent incorrect results.
- A transaction must obtain a read or write lock on a data item before it can perform a read or write operation.
- The read lock is also called a shared lock. The write lock is also known as an exclusive lock. The lock depending on its types gives or denies access to other operations on the same data item.

The basic rules for locking are,

- If a transaction has a read lock on a data item, it can read the item but not update it.
- If a transaction has a read lock on a data item, other transactions can obtain a read lock on the data item, but no write locks.
- If a transaction has a write lock on a data item, it can both read and update the data item.
- If a transaction has a write lock on a data item, then other transactions cannot obtain either a read lock or a write lock on the data item.

The locking worksas,

- All transactions that needs to access a data item must first acquire a read lock or write lock on the data item depending on whether it is a ready only operation or not.
- If the data item for which the lock is requested is not already locked, the transaction is granted the requested lock,
- If the item is currently lock, the DBMS determines what kind of lock is the current one. The DBMS also finds out what lock is requested.
- If a read lock is requested on an item that is already under a read lock, then the requested will be granted.
- If a read lock or a write lock is requested on an item that is already under a write lock, then the request is denied and the transaction must wait until the lock is released.
- A transaction continues to hold the lock until it explicitly releases it either during execution or when it terminates.
- The effects of a write operation will be visible to other transactions only after the write lock is released.

Locking Mechanisms**Explain various locking mechanisms in detail.****1. Lock-Based Protocols**

- ┆ A lock is a mechanism to control concurrent access to a data item.
- ┆ It assures that one process should not retrieve or update a record which another process is updating.
- ┆ For example, in traffic, there are signals which indicate stop and go.
- ┆ When one signal is allowed to pass at a time, then other signals are locked.
- ┆ Similarly, in database transaction only one transaction is performed at a time and other transactions are locked.
- ┆ If the locking is not done properly, then it will display the inconsistent and corrupt data.
- ┆ It manages the order between the conflicting pairs among transactions at the time of execution.

There are two lock modes,

- ┆ Binary Lock
- ┆ Shared Lock / Exclusive Lock

Binary Lock

- ┆ A lock on a data item can be in two states; it is either locked or unlocked.

Shared (S) Lock Mode

- ┆ Shared Locks are represented by S.
- ┆ The data items can only be read without performing modification to it from the database.
- S – lock is requested using lock – s instruction.

Exclusive (X) Lock Mode

- ┆ Exclusive Locks are represented by X.
- ┆ The data items can be read as well as written.

- X – lock is requested using lock – X instruction.

Lock Compatibility Matrix

- ┆ Lock Compatibility Matrix controls whether multiple transactions can acquire locks on the same resource at the same time.

	Shared	Exclusive
Shared	True	False
Exclusive	False	False

(Or)

	S	X
S	true	false
X	false	false

- ┆ If a resource is already locked by another transaction, then a new lock request can be granted only if the mode of the requested lock is compatible with the mode of the existing lock.
- ┆ Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive lock on item, no other transaction may hold any lock on the item.
- ┆ Example of a transaction performing locking:

```

T2: lock-S(A);
read(A);
unlock(A);
lock-S(B);
read(B);
unlock(B);
display(A+B)
    
```

- Locking as above is not sufficient to guarantee serializability — if A and B get updated in-between the read of A and B, the displayed sum would be wrong.
- ┆ A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

Pitfalls of Lock-Based Protocols

- ┆ Consider the partial schedule

T ₃	T ₄
lock-X(B) read(B) B := B - 50 write(B)	
	lock-S(A) read(A) lock-S(B)
lock-X(A)	

- Neither T_3 nor T_4 can make progress — executing lock-**S**(B) causes T_4 to wait for T_3 to release its lock on B , while executing lock-**X**(A) causes T_3 to wait for T_4 to release its lock on A .
- [Such a situation is called a **deadlock**.
 - ✓ To handle a deadlock one of T_3 or T_4 must be rolled back and its locks released.
- [The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil.
- [**Starvation** is also possible if concurrency control manager is badly designed. For example:
 - ✓ A transaction may be waiting for an **X**-lock on an item, while a sequence of other transactions request and are granted an **S**-lock on the same item.
 - ✓ The same transaction is repeatedly rolled back due to deadlocks.
- [Concurrency control manager can be designed to prevent starvation.

2. Timestamp Based Protocol

- [Timestamp Based Protocol helps DBMS to identify the transactions.
- [It is a unique identifier. Each transaction is issued a timestamp when it enters into the system.
- [Timestamp protocol determines the serializability order.
- [It is most commonly used concurrency protocol.
- [It uses either system time or logical counter as a timestamp.
- [It starts working as soon as a transaction is created.

- [Each transaction is issued a timestamp when it enters the system.
- [If an old transaction T_i has time-stamp $TS(T_i)$, a new transaction T_j is assigned time-stamp $TS(T_j)$ such that $TS(T_i) < TS(T_j)$.
- [The protocol manages concurrent execution such that the time-stamps determine the serializability order.
- [In order to assure such behavior, the protocol maintains for each data Q two timestamp values:
 - ✓ **W-timestamp** (Q) is the largest time-stamp of any transaction that executed **write**(Q) successfully.
 - ✓ **R-timestamp** (Q) is the largest time-stamp of any transaction that executed **read**(Q) successfully.
- [The timestamp ordering protocol ensures that any conflicting **read** and **write** operations are executed in timestamp order.
- [Suppose a transaction T_i issues a **read**(Q)
 - ✓ If $TS(T_i) \leq$ **W-timestamp** (Q), then T_i need to read a value of Q that was already overwritten.
 - Hence, the **read** operation is rejected, and T_i is rolled back.

- ✓ If $TS(T_i) \geq W\text{-timestamp}(Q)$, then the **read** operation is executed, and $R\text{-timestamp}(Q)$ is set to $\max(R\text{-timestamp}(Q), TS(T_i))$.
- [Suppose that transaction T_i issues **write**(Q).
 - ✓ If $TS(T_i) < R\text{-timestamp}(Q)$, then the value of Q that T_i is producing was needed previously, and the system assumed that that value would never be produced.
 - Hence, the **write** operation is rejected, and T_i is rolled back.
 - ✓ If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of Q .
 - Hence, this **write** operation is rejected, and T_i is rolled back.
 - ✓ Otherwise, the **write** operation is executed, and $W\text{-timestamp}(Q)$ is set to $TS(T_i)$.

Example use of the Protocol

- A partial schedule for several data items for transactions with timestamps 1, 2, 3, 4, 5.

	T_1	T_2	T_3	T_4	T_5
	read(Y)	read(Y)			read(X)
			write(Y) write(Z)		read(Z)
	read(X)	read(X) abort	write(Z) abort		write(Y) write(Z)

3. Timestamp Ordering Protocol

- ┆ The timestamp ordering Protocol ensures serializability among transactions in their conflicting read and write operations.
- ┆ This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.
- ┆ The timestamp of transaction T_i is denoted as $TS(T_i)$.
- ┆ Read time-stamp of data-item X is denoted by $R\text{-timestamp}(X)$.
- Write time-stamp of data-item X is denoted by $W\text{-timestamp}(X)$. Timestamp ordering protocol works as follows –
 - **If a transaction T_i issues a read(X) operation –**
 - If $TS(T_i) < W\text{-timestamp}(X)$
 - Operation rejected.
 - If $TS(T_i) \geq W\text{-timestamp}(X)$
 - Operation executed.
 - All data-item timestamps updated.
 - **If a transaction T_i issues a write(X) operation –**
 - If $TS(T_i) < R\text{-timestamp}(X)$
 - Operation rejected.
 - If $TS(T_i) < W\text{-timestamp}(X)$
 - Operation rejected and T_i rolled back.
 - Otherwise, operation executed.

Following are the Timestamp Ordering Algorithms,

1. Basic Timestamp Ordering

- It compares the timestamp of T with Read_TS(X) and Write_TS(X) to ensure that the transaction execution is not violated.
- If the transaction execution order is violated, transaction T is aborted and resubmitted to the system as a new transaction with a new timestamp.

2. Strict Timestamp Ordering

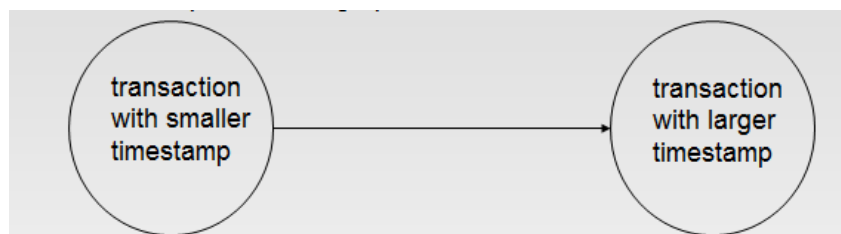
- It ensures that the schedules are both strict for easy recoverability and conflict serializability.

3. Thomas's Write Rule

- Modified version of the timestamp-ordering protocol in which obsolete **write** operations may be ignored under certain circumstances.
- It does not enforce conflict serializability.
- This rule states if $TS(T_i) < W\text{-timestamp}(X)$, then the operation is rejected and T_i is rolled back.
- It rejects some write operations, by modifying the checks for the write_item(X) operation as follows,
 - ✓ If $Read_TS(X) > TS(T)$ (read timestamp is greater than timestamp transaction), then abort and rollback transaction T and reject the operation.
 - ✓ If $Write_TS(X) > TS(T)$ (write timestamp is greater than timestamp transaction), then do not execute the write operation but continue processing. Because some transaction with a timestamp is greater than $TS(T)$ and after T in the timestamp has already written the value of X.
 - ✓ If neither the condition transaction 1 nor the condition in transaction 2 occurs, then execute the Write_item(X) operation of transaction T and set Write_TS(X) to $TS(T)$.
- Otherwise this protocol is the same as the timestamp ordering protocol.
- Thomas' Write Rule allows greater potential concurrency.
 - ✓ Allows Time-stamp ordering rules allow to make the schedule view serializable.
- Instead of making T_i rolled back, the 'write' operation itself is ignored.

Correctness of Timestamp-Ordering Protocol

- The timestamp-ordering protocol guarantees serializability since all the arcs in the precedence graph are of the form:



- Thus, there will be no cycles in the precedence graph
 - ✓ Timestamp protocol ensures freedom from deadlock as no transaction ever waits.
 - ✓ But the schedule may not be cascade-free, and may not even be recoverable.

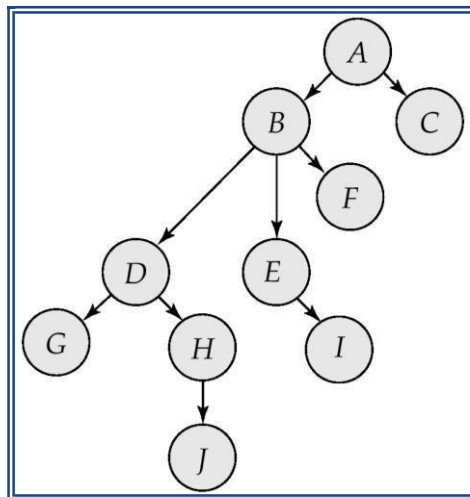
Recoverability and Cascade Freedom

- Problem with timestamp-ordering protocol:
 - ✓ Suppose T_i aborts, but T_j has read a data item written by T_i .
 - ✓ Then T_j must abort; if T_j had been allowed to commit earlier, the schedule is not recoverable.
 - ✓ Further, any transaction that has read a data item written by T_j must abort.
 - ✓ This can lead to cascading rollback --- that is, a chain of rollbacks.
- **Solution 1:**
 - ✓ A transaction is structured such that its writes are all performed at the end of its processing.
 - ✓ All writes of a transaction form an atomic action; no transaction may execute while a transaction is being written.
 - ✓ A transaction that aborts is restarted with a new timestamp.
- **Solution 2:** Limited form of locking: wait for data to be committed before reading it.
- **Solution 3:** Use commit dependencies to ensure recoverability.

4. Graph based Protocols

- Graph-based protocols are an alternative to two-phase locking
- Impose a partial ordering \rightarrow on the set $\mathbf{D} = \{d_1, d_2, \dots, d_h\}$ of all data items.
 - ✓ If $d_i \rightarrow d_j$ then any transaction accessing both d_i and d_j must access d_i before accessing d_j .
 - ✓ Implies that the set \mathbf{D} may now be viewed as a directed acyclic graph, called a *database graph*.
- The *tree-protocol* is a simple kind of graph protocol.

Tree Protocol



1. Only exclusive locks are allowed.
2. The first lock by T_i may be on any data item. Subsequently, a data Q can be locked by T_i only if the parent of Q is currently locked by T_i .
3. Data items may be unlocked at any time.

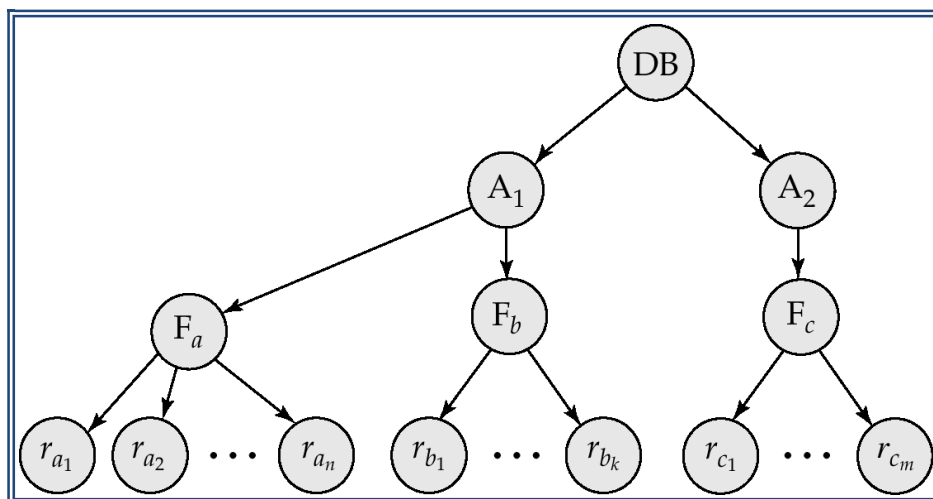
4. A data item that has been locked and unlocked by T_i cannot subsequently be relocked by T_i
- The tree protocol ensures conflict serializability as well as freedom from deadlock.
 - Unlocking may occur earlier in the tree-locking protocol than in the two-phase locking protocol.
 - ✓ shorter waiting times, and increase in concurrency
 - ✓ protocol is deadlock-free, no rollbacks are required
 - Drawbacks**
 - ✓ Protocol does not guarantee recoverability or cascade freedom
 - ✓ Need to introduce commit dependencies to ensure recoverability
 - ✓ Transactions may have to lock data items that they do not access.
 - ✓ increased locking overhead, and additional waiting time
 - ✓ potential decrease in concurrency
 - ✓ Schedules not possible under two-phase locking are possible under tree protocol, and vice versa.

Multiple Granularity

Write short notes on Multiple Granularity.

- Allow data items to be of various sizes and define a hierarchy of data granularities, where the small granularities are nested within larger ones
- Can be represented graphically as a tree (but don't confuse with tree-locking protocol)
- When a transaction locks a node in the tree *explicitly*, it *implicitly* locks the entire node's descendants in the same mode.
- Granularity of locking (level in tree where locking is done):
 - ✓ **Fine Granularity** (lower in tree): high concurrency, high locking overhead.
 - ✓ **Coarse Granularity** (higher in tree): low locking overhead, low concurrency.

Example of Granularity Hierarchy



The levels, starting from the coarsest (top) level are,

- database
- area

- file
- record

Intention Lock Modes

- In addition to S and X lock modes, there are three additional lock modes with multiple granularity:
 - ✓ **Intention-Exclusive** (IX): indicates explicit locking at a lower level with exclusive or shared locks
 - ✓ **Shared and Intention-Exclusive** (SIX): the subtree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive-mode locks.
- Intention locks allow a higher level node to be locked in S or X mode without having to check all descendent nodes.

Compatibility Matrix with Intention Lock Modes

- The compatibility matrix for all lock modes is:

	IS	IX	S	SIX	X
IS	✓	✓	✓	✓	×
IX	✓	✓	×	×	×
S	✓	×	✓	×	×
SIX	✓	×	×	×	×
X	×	×	×	×	×

Multiple Granularity Locking Scheme

- Transaction T_i can lock a node Q , using the following rules:
 - ✓ The lock compatibility matrix must be observed.
 - ✓ The root of the tree must be locked first, and may be locked in any mode.
 - ✓ A node Q can be locked by T_i in S or IS mode only if the parent of Q is currently locked by T_i in either IX or IS mode.
 - ✓ A node Q can be locked by T_i in X, SIX, or IX mode only if the parent of Q is currently locked by T_i in either IX or SIX mode.
 - ✓ T_i can lock a node only if it has not previously unlocked any node (that is, T_i is two-phase).
 - ✓ T_i can unlock a node Q only if none of the children of Q are currently locked by T_i .
- Observe that locks are acquired in root-to-leaf order, whereas they are released in leaf-to-root order.

Two Phase Locking

Explain the two phase commit and three phase commit protocols. (April/May 2015) (Or) Explain about Locking Protocols. (May/June 2016) (Or) Briefly explain about two phase commit. (May/June 2016) (Or) Illustrate two phase locking protocol with an example. (Nov/Dec 2016) (Or) What is Concurrency? Explain the two phase locking with an example. (April/May 2018)

- Two phase commit is important whenever a given transaction can interact with several independent —resource managers, each managing its own set of recoverable resources and maintaining its own recovery log.
- The two phase commit protocol guarantees that if a portion of a transaction operation cannot be committed; all changes made at the other sites participating in the transaction will be undone to maintain a consistent database state.
- For example, transaction running on an IBM mainframe that updates both an IMS database and DB2 database.
- If the transaction completes successfully, then all updates to both IMS data and DB2 data, must be committed equally. If it fails then all of its updates must be rolled back.
- Transaction has completed its database processing successfully, the system broadcast instruction it issues COMMIT, not ROLLBACK on receiving that commit request, the coordinator goes through the following two-phase process:

Phase 1: Preparation

- The coordinator sends a PREPARE TO COMMIT message to all subordinates.
- The Subordinates receive the message. Write the transaction log, using the write ahead protocol and send an acknowledgement (YES/PREPARED TO COMMIT or NO/NOT PREPARED) message to coordinator.
- The coordinator makes sure that all nodes are ready to commit, or it aborts the action.
- If all nodes are PREPARED TO COMMIT, the transaction goes to phase-2 if one or more nodes reply NO or NOT PREPARED, the coordinator broadcasts an ABORT Message to all subordinates

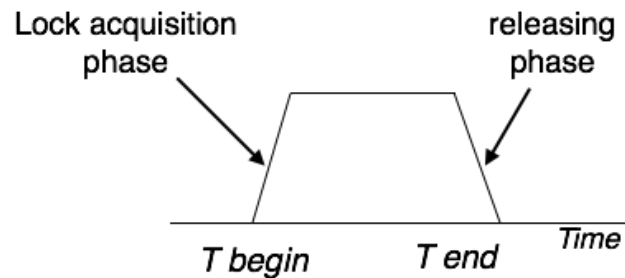
Phase 2: The Final Commit

- The coordinator broadcast a COMMIT message to all subordinates and waits for the replies
- Each subordinate receives the COMMIT message, then updates the database using the read or write operation in subordinate to coordinator.
- The subordinates reply with COMMITTED or NOT COMMITTED message to the coordinator
- If one or more subordinates did not commit, the coordinator sends an ABORT message thereby forcing them to UNDO all changes.
- The objective of the two phase commit is to ensure that all nodes commit their part of the transaction is aborted.
- If one of the nodes fails to commit, the information necessary to recover the database is in the transaction log, and the database can be recovered with do-undo-redo protocol.

The Two-Phase Locking Protocol

What is Two-Phase Locking (2PL)? Explain two phase locking protocol in detail.

- Two-Phase Locking (2PL) is a concurrency control method which divides the execution phase of a transaction into three parts.
- It ensures conflict serializable schedules.
- If read and write operations introduce the first unlock operation in the transaction, then it is said to be Two-Phase Locking Protocol.



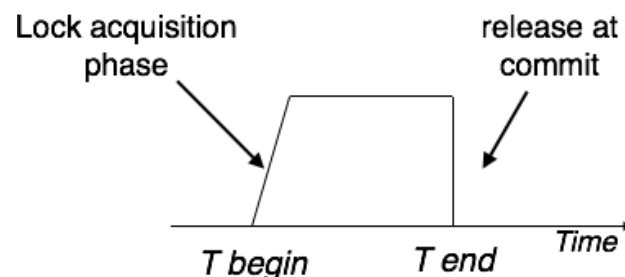
- This protocol can be divided into two phases,
 - ✓ In **Growing Phase**, a transaction obtains locks, but may not release any lock.
 - ✓ In **Shrinking Phase**, a transaction may release locks, but may not obtain any lock.
- Two-Phase Locking does not ensure freedom from deadlocks.

Types of Two – Phase Locking Protocol

- Following are the types of two – phase locking protocol:
 - ✓ Strict Two – Phase Locking Protocol
 - ✓ Rigorous Two – Phase Locking Protocol
 - ✓ Conservative Two – Phase Locking Protocol

Strict Two-Phase Locking Protocol

- Strict Two-Phase Locking Protocol avoids cascaded rollbacks.
- This protocol not only requires two-phase locking but also all exclusive-locks should be held until the transaction commits or aborts.
- It is not deadlock free.
- It ensures that if data is being modified by one transaction, then another transaction cannot read it until first transaction commits.
- Most of the database systems implement rigorous two-phase locking protocol.



Rigorous Two-Phase Locking

- Rigorous Two – Phase Locking Protocol avoids cascading rollbacks.
- This protocol requires that all the share and exclusive locks to be held until the transaction commits.

Conservative Two-Phase Locking Protocol

- Conservative Two-Phase Locking Protocol is also called as Static Two-Phase Locking Protocol.
- This protocol is almost free from deadlocks as all required items are listed in advance.
- It requires locking of all data items to access before the transaction starts.
- This is a protocol which ensures conflict-serializable schedules.

Lock Conversions

- ┆ It is a mechanism in two phase locking mechanism which allows conversion of shared lock to exclusive lock or vice versa.
- ┆ Two-phase locking with lock conversions:

First Phase:

- can acquire a lock-S on item
- can acquire a lock-X on item
- can convert a lock-S to a lock-X (upgrade)

Second Phase:

- can release a lock-S
- can release a lock-X
- can convert a lock-X to a lock-S (downgrade)
- This protocol assures serializability. But still relies on the programmer to insert the various locking instructions.

Automatic Acquisition of Locks

- A transaction T_i issues the standard read/write instruction, without explicit locking calls.
- The operation **read**(D) is processed as:

If T_i has a lock on D

then

read(D)

else begin

if necessary wait until no other

transaction has a **lock-X** on D grant

T_i a **lock-S** on D ; read(D)

end

- **write**(D) is processed as: **if** T_i

has a **lock-X** on D **then**

write(D)

else begin

if necessary wait until no other trans. has any lock on D , if T_i has a

lock-S on D

then

upgrade lock on *D* to **lock-X** else

grant *T_i* a **lock-X** on *D*

write(*D*)

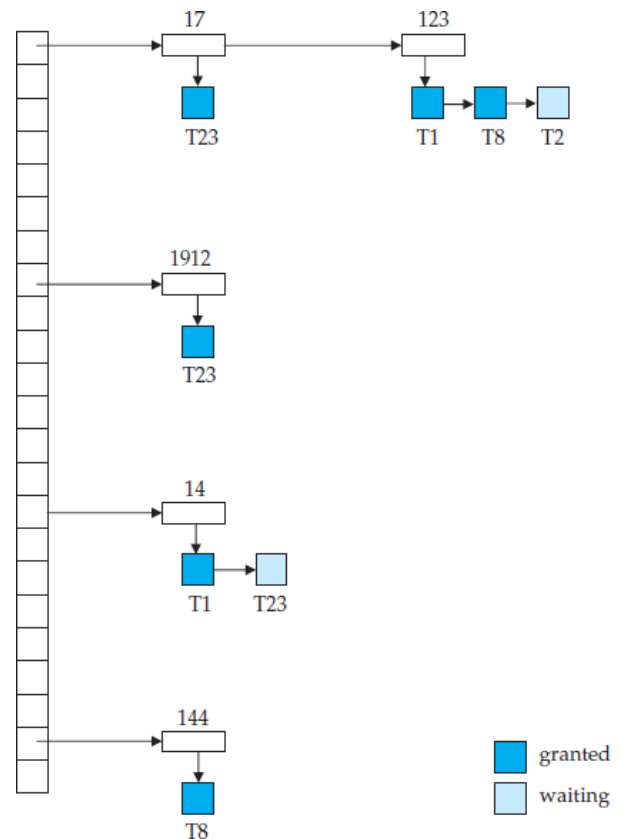
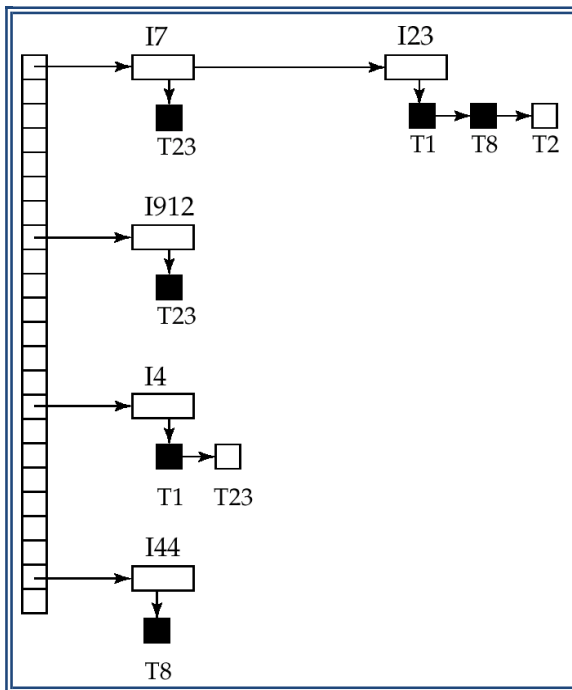
end;

- All locks are released after commit or abort

Implementation of Locking

- A **lock manager** can be implemented as a separate process to which transactions send lock and unlock requests
- The lock manager replies to a lock request by sending a lock grant message (or a message asking the transaction to roll back, in case of a deadlock)
- The requesting transaction waits until its request is answered
- The lock manager maintains a data-structure called a **lock table** to record granted locks and pending requests
- The lock table is usually implemented as an in-memory hash table indexed on the name of the data item being locked.

Lock Table



- Black rectangles indicate granted locks, white ones indicate waiting requests.
- Lock table also records the type of lock granted or requested.
- New request is added to the end of the queue of requests for the data item, and granted if it is compatible with all earlier locks.

- Unlock requests result in the request being deleted, and later requests are checked to see if they can now be granted.
- If a transaction aborts, all waiting or granted requests of that transaction are deleted.
 - ✓ Lock manager may keep a list of locks held by each transaction, to implement this efficiently.

Deadlock

Write short notes on deadlock. (Nov/Dec 2014) (Or) What is Deadlock? How does it occur? How transactions be written to avoid deadlock, guarantee correct execution? (Nov/Dec 2105) (Or) Outline deadlock handling mechanisms. (Nov/Dec 2016) (Or) When does deadlock occurs? Explain two-phase commit protocol with example. (Nov/Dec 2017) (Or) Explain the methods used to handle deadlock. (Nov/Dec 2018)

What is Deadlock?

- ┆ A deadlock is a condition that occurs when two or more different database tasks are waiting for each other and none of the task is willing to give up the resources that other task needs.
- ┆ It is an unwanted situation that may result when two or more transactions are each waiting for locks held by the other to be released.
- ┆ In deadlock situation, no task ever gets finished and is in waiting state forever.
- ┆ Deadlocks are not good for the system.

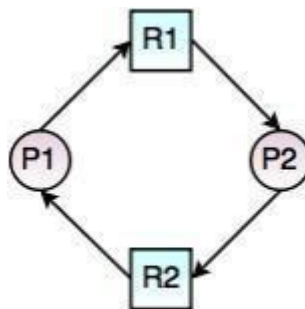


Fig. Deadlock State

In the above diagram,

- ┆ Process P1 holds Resource R2 and waits for resource R1, while Process P2 holds resource R1 and waits for Resource R2. So, the above process is in deadlock state.
- ┆ There is the only way to break a deadlock, is to abort one or more transactions.
- ┆ Once, a transaction is aborted and rolled back, all the locks held by that transaction are released and can continue their execution.
- ┆ So, the DBMS should automatically restart the aborted transactions.

Deadlock Conditions

Following are the deadlock conditions,

1. Mutual Exclusion
2. Hold and Wait
3. No Preemption
4. Circular Wait

- ☐ A deadlock may occur, if all the above conditions hold true.
 - ✓ **In Mutual exclusion states** that at least one resource cannot be used by more than one process at a time. The resource cannot be shared between processes.
 - ✓ **Hold and Wait states** that a process is holding a resource, requesting for additional resources which are being held by other processes in the system.
 - ✓ **No Preemption states** that a resource cannot be forcibly taken from a process. Only a process can release a resource that is being held by it.
 - ✓ **Circular Wait states** that one process is waiting for a resource which is being held by second process and the second process is waiting for the third process and so on and the last process is waiting for the first process. It makes a circular chain of waiting.
- ☐ There are three general techniques for handling deadlock:
 - ✓ Timeouts
 - ✓ Deadlock Prevention
 - ✓ Deadlock Detection
 - ✓ Recovery

Timeouts

- ☐ A transaction that requests a lock will wait for only a system-defined period of time.
- ☐ If the lock has not been granted within this period, the lock request times out.
- ☐ In this case, the DBMS assumes the transaction may be deadlocked, even though it may not be, and it aborts and automatically restarts the transaction.

Deadlock Prevention

- ☐ Deadlock prevention ensures that the system never enters a deadlock state. Following are the requirements to free the deadlock:
 - ☐ **No Mutual Exclusion:**
 - ✓ No Mutual Exclusion means removing all the resources that are sharable.
 - ☐ **No Hold and Wait:**
 - ✓ Removing hold and wait condition can be done if a process acquires all the resources that are needed before starting out.
 - ☐ **Allow Preemption:**
 - ✓ Allowing preemption is as good as removing mutual exclusion.
 - ✓ The only need is to restore the state of the resource for the preempted process rather than letting it in at the same time as the preemptor.
 - ☐ **Removing Circular Wait:**
 - ✓ The circular wait can be removed only if the resources are maintained in a hierarchy and a process can hold the resources in increasing the order of precedence.
- ☐ There are deadlock prevention schemes that use timestamp ordering mechanism of transactions in order to predetermine a deadlock situation.

Wait-Die Scheme

- In this scheme, if a transaction requests to lock a resource (data item), which is already held with a conflicting lock by another transaction, then one of the two possibilities may occur –
 - ✓ If $TS(T_i) < TS(T_j)$ – that is T_i , which is requesting a conflicting lock, is older than T_j – then T_i is allowed to wait until the data-item is available.
 - ✓ If $TS(T_i) > TS(T_j)$ – that is T_i is younger than T_j – then T_i dies. T_i is restarted later with a random delay but with the same timestamp.
- This scheme allows the older transaction to wait but kills the younger one.

Wound-Wait Scheme

- In this scheme, if a transaction requests to lock a resource (data item), which is already held with conflicting lock by some other transaction, one of the two possibilities may occur –
 - ✓ If $TS(T_i) < TS(T_j)$, then T_i forces T_j to be rolled back – that is T_i wounds T_j . T_j is restarted later with a random delay but with the same timestamp.
 - ✓ If $TS(T_i) > TS(T_j)$, then T_i is forced to wait until the resource is available.
- This scheme, allows the younger transaction to wait; but when an older transaction requests an item held by a younger one, the older transaction forces the younger one to abort and release the item.
- In both the cases, the transaction that enters the system at a later stage is aborted.
- Both in *wait-die* and in *wound-wait* schemes, a rolled back transaction is restarted with its original timestamp. Older transactions thus have precedence over newer ones, and starvation is hence avoided.
- Timeout-Based Schemes:
 - ✓ A transaction waits for a lock only for a specified amount of time. After that, the wait times out and the transaction is rolled back.
 - ✓ Thus deadlocks are not possible.
 - ✓ Simple to implement; but starvation is possible. Also difficult to determine good value of the timeout interval.

Deadlock Avoidance

- ┆ Deadlock Avoidance helps in avoiding the rolling back conflicting transactions.
- ┆ It is not good or practical approach to abort a transaction when a deadlock occurs.
- ┆ Instead, deadlock avoidance mechanisms can be used to detect any deadlock situation in advance.
- Deadlock occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T in the set.

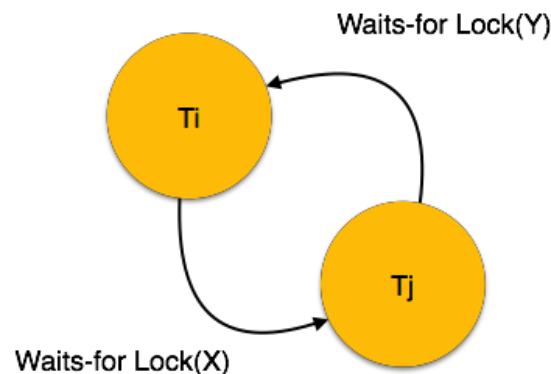
(Or)

- System is deadlocked if there is a set of transactions such that every transaction in the set is waiting for another transaction in the set.
- There is only one way to break deadlock: abort one or more of the transactions. This usually involves undoing all the changes made by the aborted transactions (S).

- ┆ Methods like "wait-for-graph" are available but they are suitable for only those systems where transactions are lightweight having fewer instances of resource.
- ┆ In a bulk system, deadlock prevention techniques may work well.

Wait-for Graph

- ┆ This is a simple method available to track if any deadlock situation may arise.
- ┆ For each transaction entering into the system, a node is created.
- ┆ When a transaction T_i requests for a lock on an item, say X , which is held by some other transaction T_j , a directed edge is created from T_i to T_j .
- ┆ If T_j releases item X , the edge between them is dropped and T_i locks the data item.
- ┆ The system maintains this wait-for graph for every transaction waiting for some data items held by others.
- ┆ The system keeps checking if there's any cycle in the graph.



Here, we can use any of the two following approaches –

- ┆ First, do not allow any request for an item, which is already locked by another transaction. This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.
- ┆ The second option is to roll back one of the transactions.
- ┆ It is not always feasible to roll back the younger transaction, as it may be important than the older one.
- ┆ With the help of some relative algorithm, a transaction is chosen, which is to be aborted.
- ┆ This transaction is known as the **victim** and the process is known as **victim selection**. **Deadlock**

Recovery

- [When deadlock is detected :
 - Some transaction will have to be rolled back (made a victim) to break deadlock. Select that transaction as victim that will incur minimum cost.
 - Rollback -- determine how far to roll back transaction
 - Total rollback: Abort the transaction and then restart it.
 - More effective to roll back transaction only as far as necessary to break deadlock.
 - Starvation happens if same transaction is always chosen as victim. Include the number of rollbacks in the cost factor to avoid starvation.

Transaction Recovery**Explain in detail about transaction recovery.**

- A transaction is the basic logical unit of execution in an information system.
- A transaction is a sequence of operations that must be executed as a whole, taking a consistent (& correct) database state into another consistent (& correct) database state.
- Data **recovery** is the process of restoring data that has been lost, accidentally deleted, corrupted or made inaccessible.

(Or)

- Data **recovery** typically refers to the restoration of data to a desktop, laptop, server or external storage system from a backup.
- The techniques used to **recover** the lost data due to system crash, **transaction** errors, viruses, catastrophic failure, incorrect commands execution etc. are called as database **recovery** techniques.

What is Recovery?

- ┆ Recovery is the process of restoring a database to the correct state in the event of a failure.
- ┆ It ensures that the database is reliable and remains in consistent state in case of a failure.

Failure Classification

- We generalize a failure into various categories, as follows
- There are some common causes of failures such as,
 1. System Crash
 2. Transaction Failure
 3. Network Failure
 4. Disk Failure
 5. Media Failure
- ┆ Each transaction has ACID property. If we fail to maintain the ACID properties, it is the failure of the database system.

1. System Crash

- ┆ System crash occurs when there is a hardware or software failure or external factors like a power failure.
- ┆ The data in the secondary memory is not affected when system crashes because the database has lots of integrity. Checkpoint prevents the loss of data from secondary memory.

2. Transaction Failure

- A transaction has to abort when it fails to execute or when it reaches a point from where it can't go any further.
- ┆ This is called transaction failure where only a few transactions or processes are hurt.
- Reasons for a transaction failure could be –
- **Logical errors** – Where a transaction cannot complete because it has some code error or any internal error condition.

- **System errors** – Where the database system itself terminates an active transaction because the DBMS is not able to execute it, or it has to stop because of some system condition. For example, in case of deadlock or resource unavailability, the system aborts an active transaction.
- ┆ This failure occurs when there are system errors like deadlock or unavailability of system resources to execute the transaction.

3. Network Failure

- A network failure occurs when a client – server configuration or distributed database system are connected by communication networks.

4. Disk Failure

- ┆ Disk Failure occurs when there are issues with hard disks like formation of bad sectors, disk head crash, unavailability of disk etc.

5. Media Failure

- ┆ Media failure is the most dangerous failure because, it takes more time to recover than any other kind of failures.
- ┆ A disk controller or disk head crash is a typical example of media failure.
- ┆ Natural disasters like floods, earthquakes, power failures, etc. damage the data.

Storage Structure / Storage of Data

- ┆ A DBMS stores the data on external storage because the amount of data is very huge and must persist across program executions.
- ┆ Data storage is the memory structure in the system.
- The storage structure/storage of data can be divided into three categories –
 - ☐ Volatile Memory
 - ✓ Non – Volatile Memory
 - ☐ Stable Memory

Volatile Memory

- ┆ Volatile memory can store only a small amount of data. For eg. Main memory, cache memory etc.
- ┆ Volatile memory is the primary memory device in the system and placed along with the CPU.
- ┆ In volatile memory, if the system crashes, then the data will be lost.
- ┆ RAM is a primary storage device which stores a disk buffer, active logs and other related data of a database.
- ┆ Primary memory is always faster than secondary memory.
- ┆ When we fire a query, the database fetches a data from the primary memory and then moves to the secondary memory to fetch the record.
- ┆ If the primary memory crashes, then the whole data in the primary memory is lost and cannot be recovered.
- ┆ To avoid data loss, create a copy of primary memory in the database with all the logs and buffers, create checkpoints at several places so the data is copied to the database.

Non - Volatile Memory

- Non – volatile memory is the secondary memory.
- ┆ These memories are huge in size, but slow in processing. For eg. Flash memory, hard disk, magnetic tapes etc.
- ┆ If the secondary memory crashes, whole data in the primary memory is lost and cannot be recovered.

To avoid data loss in the secondary memory, there are three methods used to back it up:

1. Remote backup creates a database copy and stores it in the remote network.
 - ✓ The database is updated with the current database and sync with data and other details.
 - ✓ The remote backup is also called as an offline backup because it can be updated manually.
 - ✓ If the current database fails, then the system automatically switches to the remote database and starts functioning. The user will not know that there was a failure.
2. The database is copied to secondary memory devices like Flash memory, hard disk, magnetic tapes, etc. and kept in a secured place.
 - ✓ If the system crashes or any failure occurs, the data would be copied from these tapes to bring the database up.
3. The huge amount of data is an overhead to backup the whole database.
 - ✓ To overcome this problem the log files are backed up at regular intervals. The log file includes all the information about the transaction being made.
 - ✓ These files are backed up at regular intervals and the database is backed up once in a week.

Stable Memory

- ┆ Stable memory is the third form of the memory structure and same as non-volatile memory.
- In stable memory, copies of the same non – volatile memories are stored in different places, because if the system crashes and data loss occurs, the data can be recovered from other copies.

Log-Based Recovery

- ┆ Logs are the sequence of records that maintain the records of actions performed by a transaction.
- In Log–Based Recovery, log of each transaction is maintained in some stable storage. If any failure occurs, it can be recovered from there to recover the database.
- ┆ The log contains the information about the transaction being executed, values that have been modified and transaction state.
- ┆ All these information will be stored in the order of execution.

Log-based recovery works as follows –

- ┆ The log file is kept on a stable storage media.
- ┆ When a transaction enters the system and starts execution, it writes a log about it.

<T_n, Start>

- When the transaction modifies an item X, it write logs as follows –
 $\langle T_n, X, V_1, V_2 \rangle$
- [It reads T_n has changed the value of X, from V_1 to V_2 .
- When the transaction finishes, it logs –
 $\langle T_n, \text{commit} \rangle$
- [The database can be modified using two approaches. (Or) There are two methods of creating the log files and updating the database,
 1. Deferred Database Modification
 2. Immediate Database Modification

Deferred Database Modification

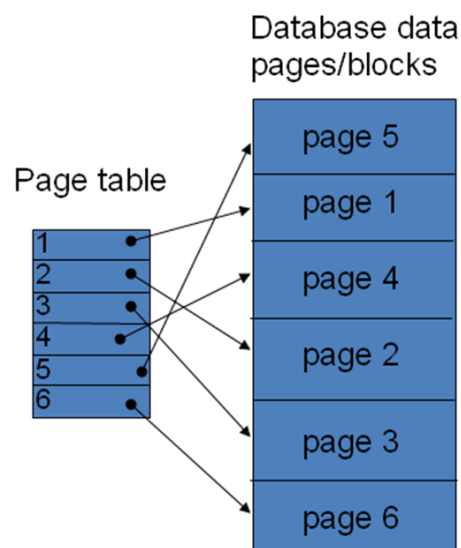
- [All the logs for the transaction are created and stored in some storage system.
- [In the above example, three log records are created and stored in some storage system, the database will be updated with those steps.

Immediate Database Modification

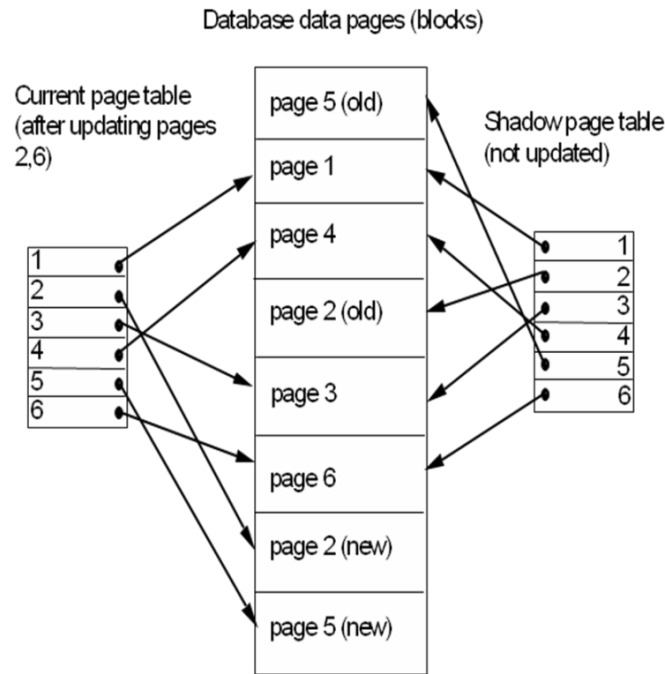
- [After creating each log record, the database is modified for each step of log entry immediately.
- [In the above example, the database is modified at each step of log entry that means after first log entry, transaction will hit the database to fetch the record, then the second log will be entered followed by updating the employee's address, then the third log followed by committing the database changes.

Shadow Paging Technique

- Data isn't updated 'in place'
- [The database is considered to be made up of a number of n fixed-size disk blocks or pages, for recovery purposes.
- [A page table with n entries is constructed where the i^{th} page table entry points to the i^{th} database page on disk.
- [Current page table points to most recent current database pages on disk



- When a transaction begins executing,
 - the current page table is copied into a shadow page table
 - shadow page table is then saved
 - shadowpage table is never modified during transaction execution
 - ✓ writes operations—new copy of database page is created and current page table entry modified to point to new disk page/block



- To recover from a failure
 - ✓ the state of the database before transaction execution is available through the shadow page table
 - ✓ free modified pages
 - ✓ discard current page table
 - ✓ that state is recovered by reinstating the shadow page table to become the current page table once more
- Committing a transaction
 - ✓ discard previous shadow page
 - ✓ free old page tables that it references
- Garbage Collection

Recovery with Concurrent Transaction

- ┆ When more than one transaction is executed in parallel, the logs are interleaved.
- ┆ At that time of recovery, it would become difficult for the recovery system to return all logs to a previous point and then start recovering.
- ┆ To overcome this situation 'Checkpoint' is used.

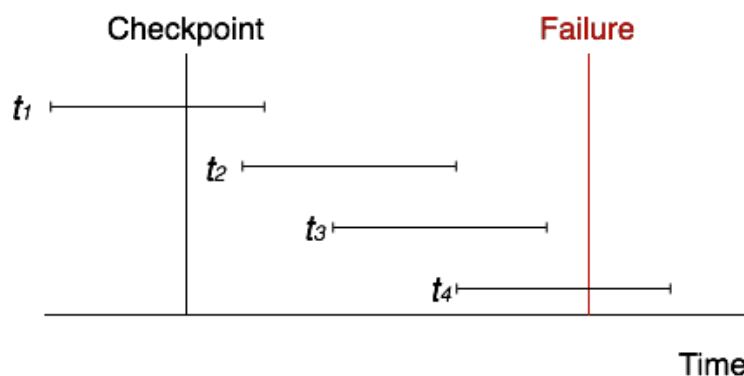
Checkpoint

- ┆ Checkpoint acts like a benchmark.
- ┆ Checkpoints are also called as **Synpoints** or **Savepoints**.

- ┆ It is a mechanism where all the previous logs are removed from the system and stored permanently in a storage system.
- ┆ It declares a point before which the database management system was in consistent state and all the transactions were committed.
- ┆ It is a point of synchronization between the database and the transaction log file.
- ┆ It involves operations like writing log records in main memory to secondary storage, writing the modified blocks in the database buffers to secondary storage and writing a checkpoint record to the log file.
- ┆ The checkpoint record contains the identifiers of all transactions that are active at the time of the checkpoint.

Recovery

- ┆ When concurrent transactions crash and recover, the checkpoint is added to the transaction and recovery system recovers the database from failure in following manner,
 - When a system with concurrent transactions crashes and recovers, it behaves in the following manner –
 - ┆ The recovery system reads the logs backwards from the end to the last checkpoint.
 - ┆ It maintains two lists, an undo-list and a redo-list.
 - ┆ If the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ and $\langle T_n, \text{Commit} \rangle$ or just $\langle T_n, \text{Commit} \rangle$, it puts the transaction in the redo-list.
 - ┆ If the recovery system sees a log with $\langle T_n, \text{Start} \rangle$ but no commit or abort log found, it puts the transaction in undo-list.
- ┆ All the transactions in the undo log are undone and their logs are removed.
- ┆ All the transactions in the redo log and their previous logs are removed and then redone before saving their logs.



Save Points

Write short notes on save points.

- ┆ A savepoint is a way of implementing subtransactions (also known as nested transactions) within a relational **database management system** by indicating a **point** within a transaction that can be "rolled back to" without affecting any work done in the transaction before the savepoint was created.
- ┆ Multiple savepoints can exist within a single transaction.
- ┆ Savepoints are useful for implementing complex error recovery in database applications.

- ┆ If an error occurs in the midst of a multiple-statement transaction, the application may be able to recover from the error (by rolling back to a savepoint) without needing to abort the entire transaction.
- ┆ A savepoint can be declared by issuing a **SAVEPOINT name** statement.
- ┆ All changes made after a savepoint has been declared can be undone by issuing a **ROLLBACK TO SAVEPOINT name** command.
- ┆ Issuing **RELEASE SAVEPOINT name** will cause the named savepoint to be discarded, but will not otherwise affect anything.
- ┆ Issuing the commands **ROLLBACK** or **COMMIT** will also discard any savepoints created since the start of the main transaction.

COMMIT Command

- ┆ **COMMIT** command saves all the work done.
- ┆ It ends the current transaction and makes permanent changes during the transaction.

Syntax:

commit;

SAVEPOINT Command

- ┆ **SAVEPOINT** command is used for saving all the current point in the processing of a transaction.
- ┆ It marks and saves the current point in the processing of a transaction.

Syntax:

SAVEPOINT <savepoint_name>

Example:

SAVEPOINT no_update;

- ┆ It is used to temporarily save a transaction, so that you can rollback to that point whenever necessary.

ROLLBACK Command

- ┆ **ROLLBACK** command restores database to original since the last **COMMIT**.
- ┆ It is used to restore the database to last committed state.

Syntax:

ROLLBACK TO SAVEPOINT <savepoint_name>;

Example:

ROLLBACK TO SAVEPOINT no_update;

Isolation Levels

Discuss isolation levels in detail.

- Isolation level is nothing but locking the row while performing some task, so that other transaction cannot access or will wait for the current transaction to finish its job.
- ┆ Serializability is a useful concept because it allows programmers to ignore issues related to concurrency when they code transactions.
- ┆ If every transaction has the property that it maintains database consistency if executed alone, then serializability ensures that concurrent executions maintain consistency.

- ┆ The SQL standard also allows a transaction to specify that it may be executed in such a way that it becomes nonserializable with respect to other transactions.

Let's write a transaction without Isolation level.

```
BEGIN TRANSACTION MyTransaction
```

```
BEGIN TRY
```

```
UPDATE Account SET Debit=100 WHERE Name='John Cena'
```

```
UPDATE ContactInformation SET Mobile='1234567890' WHERE Name='The Rock'
```

```
COMMIT TRANSACTION MyTransaction
```

```
PRINT 'TRANSACTION SUCCESS'
```

```
END TRY
```

```
BEGIN CATCH
```

```
ROLLBACK TRANSACTION MyTransaction
```

```
PRINT 'TRANSACTION FAILED'
```

```
END CATCH
```

- ┆ SQL Server provides 5 Isolation levels to implement with SQL Transaction to maintain data concurrency in the database. The isolation levels specified by the SQL standard are as follows:

Read uncommitted

- ┆ It allows uncommitted data to be read. It is the lowest isolation level allowed by SQL.
- ┆ All the isolation levels above additionally disallow **dirty writes**, that is, they disallow writing to a data item that has already been written by another transaction that has not yet committed or aborted.

Example

```
SET TRANSACTION ISOLATION LEVEL
```

```
READ UNCOMMITTED
```

```
BEGIN TRANSACTION MyTransaction
```

```
BEGIN TRY
```

```
UPDATE Account SET Debit=100 WHERE Name='John Cena'
```

```
UPDATE ContactInformation SET Mobile='1234567890' WHERE Name='TheRock'
```

```
COMMIT TRANSACTION MyTransaction
```

```
PRINT 'TRANSACTION SUCCESS'
```

```
END TRY
```

```
BEGIN CATCH
```

```
ROLLBACK TRANSACTION MyTransaction
```

```
PRINT 'TRANSACTION FAILED'
```

```
END CATCH
```

Read committed

- ┆ It allows only committed data to be read, but does not require repeatable reads.
- ┆ For instance, between two reads of a data item by the transaction, another transaction may have updated the data item and committed.

Example

```

SET TRANSACTION ISOLATION LEVEL
READ COMMITTED
BEGIN TRANSACTION MyTransaction
BEGIN TRY
UPDATE Account SET Debit=100 WHERE Name='John Cena'
UPDATE ContactInformation SET Mobile='1234567890' WHERE Name='TheRock'
COMMIT TRANSACTION MyTransaction
PRINT 'TRANSACTION SUCCESS'
END TRY
BEGIN CATCH
ROLLBACK TRANSACTION MyTransaction
PRINT 'TRANSACTION FAILED'
END CATCH

```

Repeatable read

- ┆ It allows only committed data to be read and further requires that, between two reads of a data item by a transaction, no other transaction is allowed to update it.
- ┆ However, the transaction may not be serializable with respect to other transactions.
- ┆ For instance, when it is searching for data satisfying some conditions, a transaction may find some of the data inserted by a committed transaction, but may not find other data inserted by the same transaction.

Example

```

SET TRANSACTION ISOLATION LEVEL
REPEATABLE READ

```

Serializable

- ┆ It usually ensures serializable execution.
- ┆ However, as we shall explain shortly, some database systems implement this isolation level in a manner that may, in certain cases, allow nonserializable executions.

Example

```

SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE

```

Snapshot

- ┆ Snapshot Isolation (SI) is an optimistic isolation level.
- ┆ Allows for processes to read older versions of committed data if the current version is locked.
- ┆ Difference between snapshot and read committed has to do with how old the older versions have to be.
- It's possible to have two transactions executing simultaneously that give us a result that is not possible in any serial execution.

Example**SETTRANSACTIONISOLATIONLEVEL
SNAPSHOT****SQL Facilities for Concurrency and Recovery**

It is possible to achieve concurrency control and recovery using SQL statements. Following are the different types of isolations available in SQL Server.

- ┆ READ COMMITTED
- ┆ READ UNCOMMITTED
- ┆ REPEATABLE READ
- ┆ SERIALIZABLE
- ┆ SNAPSHOT

Let us discuss about each isolation level in details. Before this, execute following script to create table and insert some data that we are going to use in examples for each isolation

```
IF OBJECT_ID('Emp') is not null begin
DROP TABLE Emp
end
create table Emp(ID int,Name Varchar(50),Salary Int) insert into
Emp(ID,Name,Salary) values ( 1,'David',1000) insert into
Emp(ID,Name,Salary) values ( 2,'Steve',2000) insert into
Emp(ID,Name,Salary)values(3,'Chris',3000)
```

	ID	Name	Salary
1	1	David	1000
2	2	Steve	2000
3	3	Chris	3000

Note: Before executing each example in this article, reset the Emp table values by executing the above script.

Read Committed

- ┆ In select query it will take only committed values of table.
- ┆ If any transaction is opened and incompleted on table in others sessions then select query will wait till no transactions are pending on same table.
- ┆ Read Committed is the default transaction isolation level.

Read committed example 1:**Session 1**

```
begin tran
update emp set Salary=999 where ID=1 waitfor delay
'00:00:15'
commit
```

Session 2

```
settransactionisolationlevelreadcommitted select
```

```
Salary from Empwhere ID=1
```

- [Run both sessions side by side.

Output

999

- [In second session, it returns the result only after execution of complete transaction in first session because of the lock on Emp table.
- [We have used wait command to delay 15 seconds after updating the Emp table in transaction.

Read committed example 2

Session 1

```
begin tran
```

```
select * from Emp waitfor
```

```
delay '00:00:15' commit
```

Session 2

```
settransactionisolationlevelreadcommitted select *
```

```
fromEmp
```

- [Run both sessions side by side.

Output

1000

- [In session 2, there won't be any delay in execution because in session 1 Emp table is used under transaction but it is not used update or delete command hence Emp table is not locked.

Read committed example 3

Session 1

```
begin tran
```

```
select * from emp waitfor
```

```
delay '00:00:15'
```

```
update emp set Salary=999 where ID=1 commit
```

Session 2

```
settransactionisolationlevelreadcommitted select
```

```
Salary from Empwhere ID=1
```

- [Run both sessions side by side.

Output

1000

- In session 2, there won't be any delay in execution because when session 2 is executed Emp table in session 1 is not locked (used only select command, locking on Emp table occurs after wait delay command).

Read Uncommitted

- If any table is updated (insert or update or delete) under a transaction and same transaction is not completed that is not committed or roll backed then uncommitted values will display (Dirty Read) in select query of "Read Uncommitted" isolation transactions sessions.
- There won't be any delay in select query execution because this transaction level does not wait for committed values on table.

Read uncommitted example 1

Session 1

```
begin tran
update emp set Salary=999 where ID=1 waitfor delay
'00:00:15'
rollback
```

Session 2

```
set transaction isolation level read uncommitted select Salary
from Emp where ID=1
```

- Run both sessions at a time one by one.

Output

999

- Select query in Session 2 executes after update Emp table in transaction and before transaction rolled back.
- Hence 999 is returned instead of 1000.
- If you want to maintain Isolation level "Read Committed" but you want dirty read values for specific table then use **with (nolock)** in select query for same tables as shown below.

```
set transaction isolation level read committed select *
from Emp with(nolock)
```

Repeatable Read

- select query data of table that is used under transaction of isolation level "Repeatable Read" cannot be modified from any other sessions till transaction is completed.

Repeatable Read Example 1

Session 1

```
set transaction isolation level repeatable read begin tran
select * from emp where ID in(1,2) waitfor
delay '00:00:15'
select * from Emp where ID in(1,2)
rollback
```

Session 2

update emp set Salary=999 where ID=1

- Run both sessions side by side.

Output

- Update command in session 2 will wait till session 1 transaction is completed because emp table row with ID=1 has locked in session 1 transaction.

Repeatable Read Example 2**Session 1**

set transaction isolation level repeatable read begin tran

select * from emp waitfor

delay '00:00:15' select * from

Emp rollback

Session 2

insert into Emp(ID,Name,Salary) values(
11,'Stewart',11000)

- Run both sessions side by side.

Output

Result in Session 1.

ID	Name	Salary
1	David	1000
2	Steve	2000
3	Chris	3000

ID	Name	Salary
1	David	1000
2	Steve	2000
3	Chris	3000
11	Stewart	11000

- Session 2 will execute without any delay because it has insert query for new entry. This isolation level allows inserting new data but does not allow modifying data that is used in select query executed in transaction.
- You can notice two results displayed in Session 1 have different number of row count (1 row extra in second result set).

Repeatable Read Example 3**Session 1**

set transaction isolation level repeatable read begin tran

select * from emp where ID in(1,2)

```
waitfor delay'00:00:15'
select * from Emp where ID in(1,2)
rollback
```

Session 2

```
update emp set Salary=999 where ID=3
Run both sessions at a time one by one.
```

Output

- [Session 2 will execute without any delay because row with ID=3 is not locked, that is only 2 records whose IDs are 1, 2 are locked in Session 1.

Serializable

- [Serializable Isolation is similar to Repeatable Read Isolation but the difference is it prevents Phantom Read.
- [This works based on range lock. If table has index then it locks records based on index range used in WHERE clause (like where ID between 1 and 3).
- [If table doesn't have index then it locks complete table.

Serializable Example 1

- [Assume table does not have index column.

Session 1

```
set transaction isolation level serializable
begin tran
select * from emp
waitfor delay '00:00:15'
select * from Emp
rollback
```

Session 2

```
insert into Emp(ID,Name,Salary) values ( 11,'Stewart',11000)
```

- [Run both sessions side by side.

Output

Result in Session 1.

ID	Name	Salary
1	David	1000
2	Steve	2000
3	Chris	3000

ID	Name	Salary
1	David	1000
2	Steve	2000
3	Chris	3000

- [Complete Emp table will be locked during the transaction in Session 1.

- ☐ Unlike "Repeatable Read", insert query in Session 2 will wait till session 1 execution is completed.
- ☐ Hence Phantom read is prevented and both queries in session 1 will display same number of rows.
- ☐ To compare same scenario with "Repeatable Read" read **Repeatable Read Example 2. Serializable**

Example 2

- ☐ Assume table has primary key on column "ID".
- ☐ In our example script, primary key is not added. Add primary key on column Emp.ID before executing below examples.

Session 1

```
set transaction isolation level serializable begin tran
select * from emp where ID between 1 and 3 waitfor
delay '00:00:15'
select * from Emp where ID between 1 and 3 rollback
```

Session 2

```
insert into Emp(ID,Name,Salary) values ( 11,'Stewart',11000)
```

- ☐ Run both sessions side by side.

Output

- ☐ Since Session 1 is filtering IDs between 1 and 3, only those records whose IDs range between 1 and 3 will be locked and these records cannot be modified and no new records with ID range between 1 to 3 will be inserted.
- ☐ In this example, new record with ID=11 will be inserted in Session 2 without any delay. Snapshot
- ☐ Snapshot isolation is similar to Serializable isolation.
- ☐ The difference is Snapshot does not hold lock on table during the transaction so table can be modified in other sessions.
- ☐ Snapshot isolation maintains versioning in Tempdb for old data in case of any data modification occurs in other sessions then existing transaction displays the old data from Tempdb.

Snapshot Example 1

Session 1

```
set transaction isolation level snapshot begin tran
select * from Emp waitfor
delay '00:00:15' select * from
Emp rollback
```

Session 2

insert into Emp(ID,Name,Salary) values (11,'Stewart',11000) update Emp

set Salary=4444 where ID=4

select * from Emp

Run both sessions side by side.

Output

ResultinSession1.

	ID	Name	Salary
1	1	David	1000
2	2	Steve	2000
3	3	Chris	3000

	ID	Name	Salary
1	1	David	1000
2	2	Steve	2000
3	3	Chris	3000

ResultinSession2.

	ID	Name	Salary
1	1	David	1000
2	2	Steve	2000
3	3	Chris	3000
4	11	Stewart	11000

- Session 2 queries will be executed in parallel as transaction in session 1 won't lock the table Emp.

Unit – II

Database Design

Entity-Relationship Model - E-R Diagrams - Enhanced-ER Model - ER-to-Relational Mapping - Functional Dependencies - Non-loss Decomposition - First, Second, Third Normal Forms, Dependency Preservation - Boyce / Codd Normal Form - Multi-valued Dependencies and Fourth Normal Form - Join Dependencies and Fifth Normal Form.

Entity-Relationship Model

Explain in detail about Entity Relationship Model.

- The entity-relationship (E-R) data model uses a collection of basic objects, called *entities* and *relationships* among these objects.
- An entity is a —thing‖ or —object‖ in the real world that is distinguishable from other objects.
- For example, each person is an entity and bank accounts can be considered as entities.
- The E-R models employ the basic concepts such as,
 - ✓ Entity Sets
 - ✓ Relationship Sets
 - ✓ Attributes

Entity Set

- The set of all entities of the same type are termed as an **entity set**.
- Entities are described in a database by a set of **attributes**.
- The extra attribute *ID* is used to identify an instructor uniquely.

Types of Entity Set

- Strong Entity Set**
 - ✓ An entity set that has a primary key is called strong entity set.
- Weak Entity Set**
 - ✓ An entity set that does not have a primary key is called weak entity set.
- Identifying Or Owner Entity set**
 - ✓ Weak entity set must be associated with another entity set called identifying or owner entity set.
 - ✓ The weak entity set is said to be existence dependent on identifying entity set.

Relationship Set

- A **relationship** is an association among several entities.
- The set of all relationships of the same type are termed as a **relationship set**.

Attributes

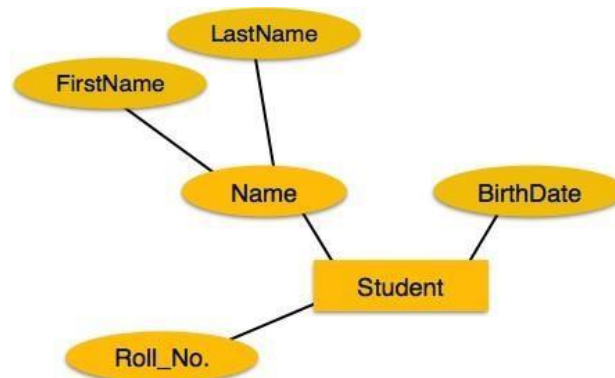
- An attribute of an entity set is a function that maps from the entity set into a domain.
- For each attributes, there is a set of permitted value set of that attribute.

Types of Attributes

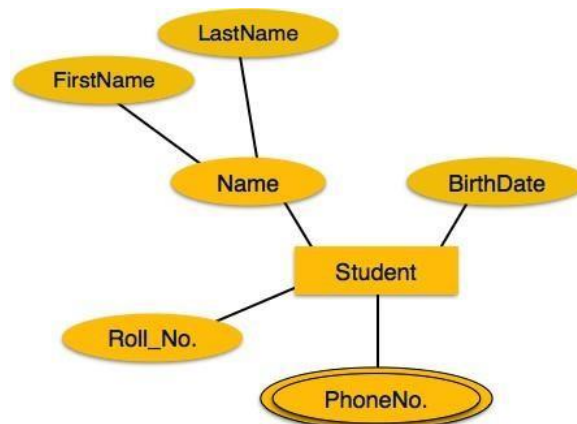
- Simple and Composite
- Single Value and Multi-valued
- Derived
- Descriptive

Simple & Composite Attributes

- Simple Attribute**
 - ✓ Attributes that cannot be divided into subparts are called simple attributes.
 - ✓ Ex: S.no is a simple attributes.
- Composite Attribute**
 - ✓ Attribute that can be divided into subparts is called as composite attributes.

**Single Valued & Multi-valued Attribute**

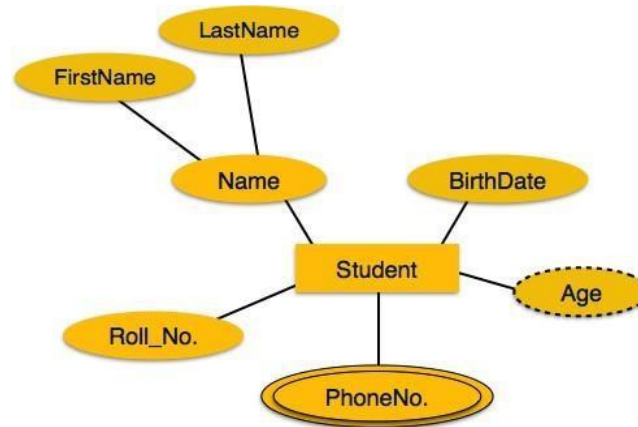
- Single Valued Attribute**
 - ✓ The attribute that have single value for a particular entity is called single valued attribute.
 - ✓ Ex: student_id – refers to only one student.
- Multi-valued Attributes**
 - ✓ The attribute that has a set of values for a specific entity is called multi-valued attributes.
 - ✓ Ex: phone_number

**Derived Attribute**

- The value of this type of attribute can be derived from the values of other related

attributes or entities.

Example:



Descriptive Attribute

- The attribute present in the relationship is called as descriptive attribute.



Figure: ER Diagram with Descriptive Attribute

□ **NULL Value:**

- ✓ The attribute takes a NULL value, when an entity does not have a value for it.

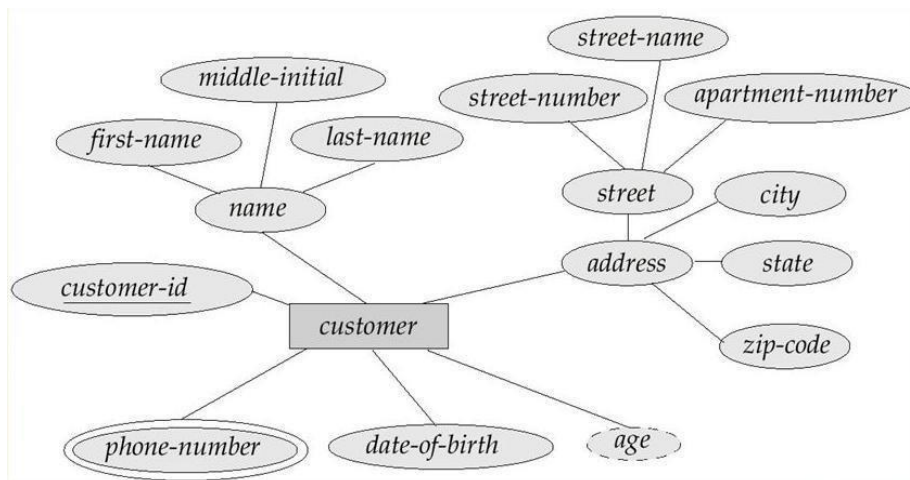


Figure: ER diagram with Composite, Multi-valued and Derived Attributes

In the above ER diagram,

- ✓ C_name & address - Composite Attribute
- ✓ Street - Component Attribute
- ✓ Phone-number - Multi-valued Attribute
- ✓ Age - Derived Attribute

Constraints**Explain in detail about constraints.**

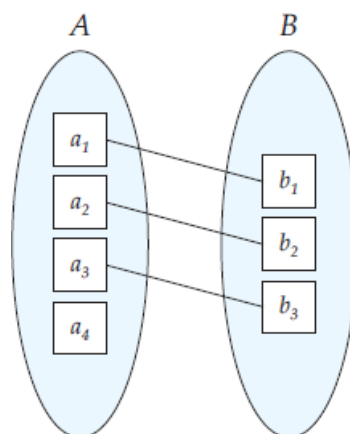
- An E-R enterprise schema may define certain constraints to which the contents of a database must conform.
- That limits the possible combinations of entities that may participate in the corresponding relationship set.
 - ✓ Mapping Cardinalities (Cardinality Constraints)
 - ✓ Participation Constraints

Mapping Cardinalities (May/June 2013)

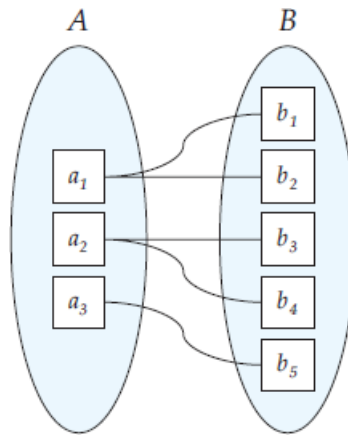
- Mapping cardinalities**, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.
- Mapping cardinalities are most useful in describing binary relationship sets, although they can contribute to the description of relationship sets that involve more than two entity sets.
- A mapping cardinality is a data constraint that specifies how many entities an entity can be related to in a relationship set.
- Consider a binary relationship set R between entity sets A and B . There are four possible mapping cardinalities in this case:
- Types of mapping cardinalities are,
 - ✓ One to One
 - ✓ One to Many
 - ✓ Many to One
 - ✓ Many to Many

One-to-One

- An entity in A is associated (related) with *at most* one entity in B , and an entity in B is associated with *at most* one entity in A .

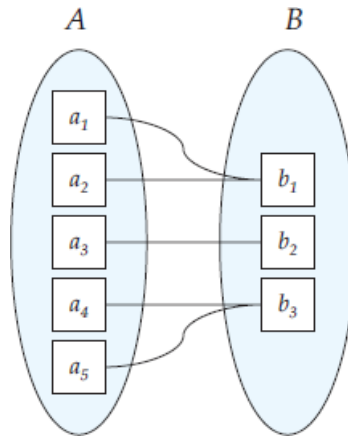
**One-to-Many**

- An entity in A is associated with any number (zero or more) of entities in B .
- And an entity in B , can be associated with *at most* one entity in A .



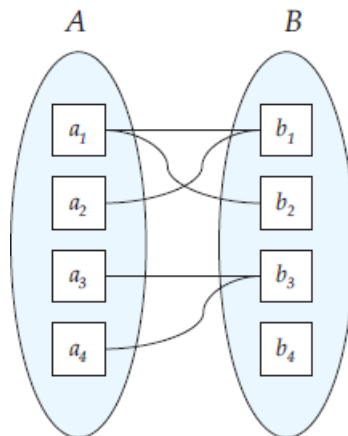
Many-to-One

- An entity in *A* is associated with *at most* one entity in *B*.
- And an entity in *B*, can be associated with any number (zero or more) of entities in *A*.



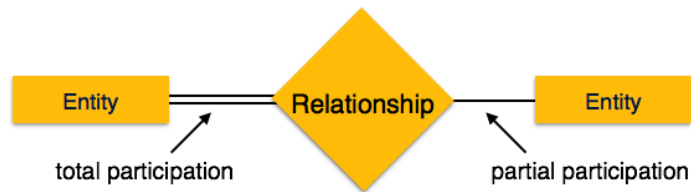
Many-to-Many

- An entity in *A* is associated with any number (zero or more) of entities in *B*, and an entity in *B* is associated with any number (zero or more) of entities in *A*.



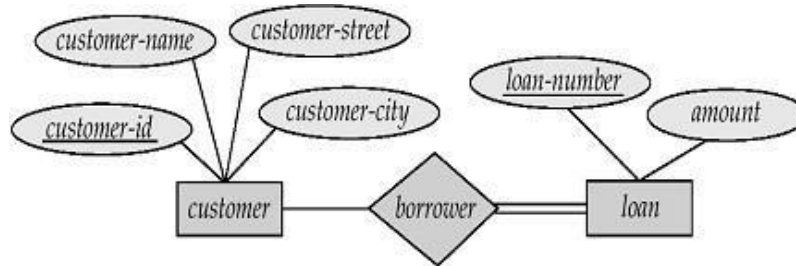
Participation Constraints

- It specifies whether the existence of an entity depends on its being related to another entity via the relationship type.



Total Participation

- The participation of an entity set E in a relationship set R is said to be total, if every entity in E participates in at least one relationship in R .



- Double line indicates that from loan to borrower, each loan must have at least one associated customer.

Partial Participation

- If only some entities in E participate in relationship in R , then the participation of entity E in relationship R is said to be partial.



Weak Entity Set

- Entity types that do not have key attributes of their own are called weak entity types.
- A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship, because a weak entity cannot be identified without an owner entity type.
- A weak entity set is indicated in E-R diagrams by a doubly outlined rectangular box and the corresponding identifying relationship by a doubly outlined diamond.

Strong Entity Set

- Entity types that have key attributes of their own are called strong entity types.
- A strong entity set is indicated in E-R diagrams by rectangular box and its relationship by a diamond.

Diagrams

- E-R diagram is used to express the overall logical structure (schema) of a database graphically by an *entity-relationship (E-R) diagram*.
- E-R diagrams are simple and clear.

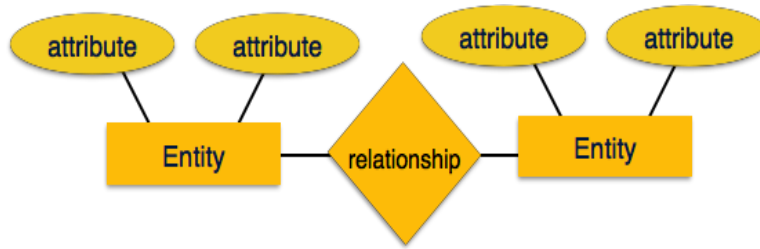
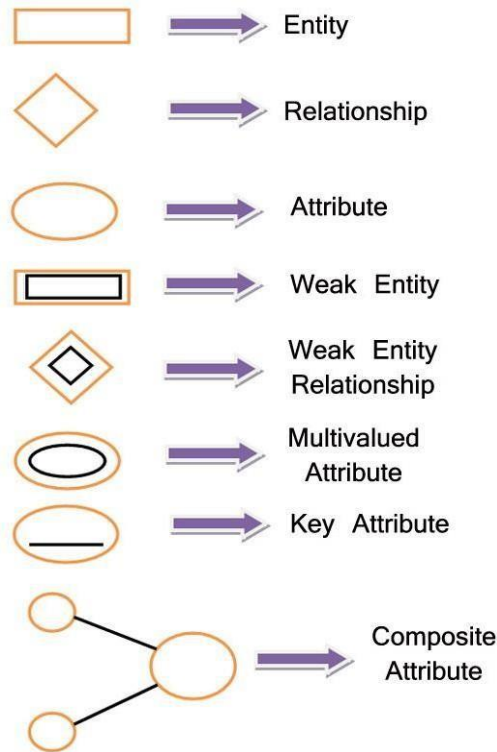


Figure: Sample ER Diagram

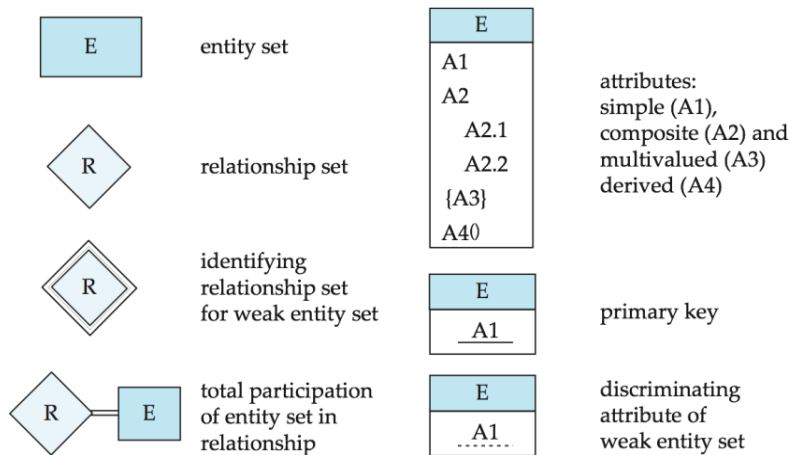
ER Diagram Representation Symbols

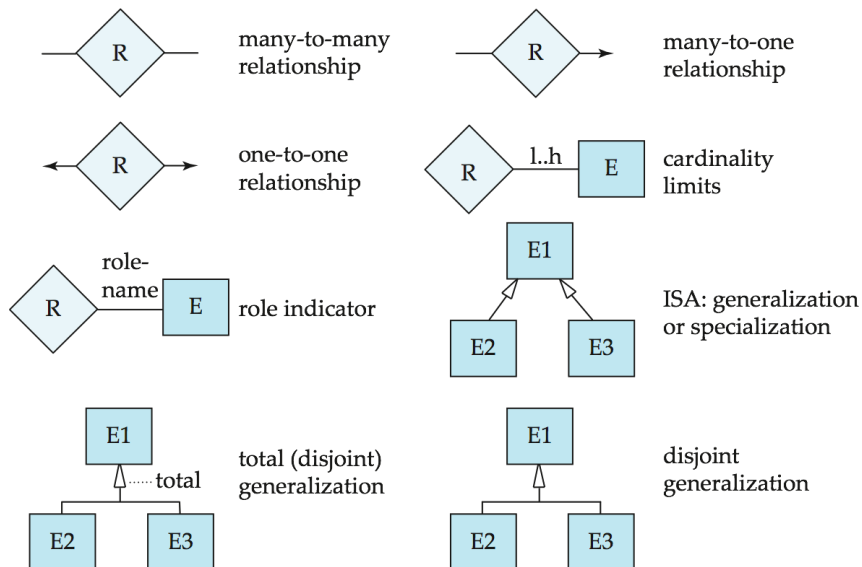
Write short notes on ER diagram representation symbols.



(Or)

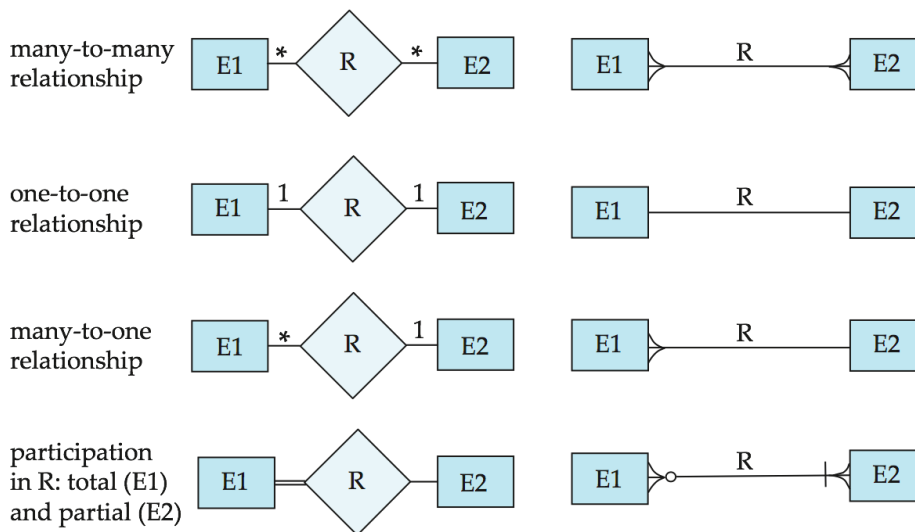
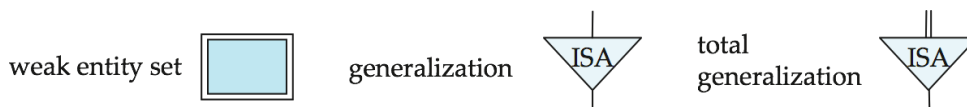
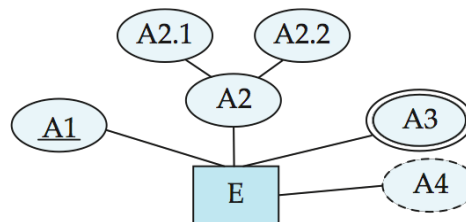
Symbols Used in E-R Notation





Alternative ER Notations

entity set E with simple attribute A1, composite attribute A2, multivalued attribute A3, derived attribute A4, and primary key A1



Basic Structure

An E-R diagram consists of the following major components:

- **Rectangles** divided into **two parts** represent entity sets.
- The first part, which in this textbook is shaded blue, contains the name of the entity set.
- The second part contains the names of all the attributes of the entity set.
- Rectangle - to represent an entity set
- Ellipses - to represent an attribute
- Diamonds - to represent relationship sets

- Lines - to link attributes to entity sets and entity sets to relationship sets
- Double ellipses - to represent multi-valued attributes
- Dashed ellipses - to represent derived attributes
- Double rectangle - to represent weak entity sets
- Double line - to represent total participation of an entity in a relationship set.
- **Double diamonds** - represent identifying relationship sets linked to weak entity sets
- **Undivided rectangles** - represent the attributes of a relationship set. Attributes that are part of the primary key are underlined.

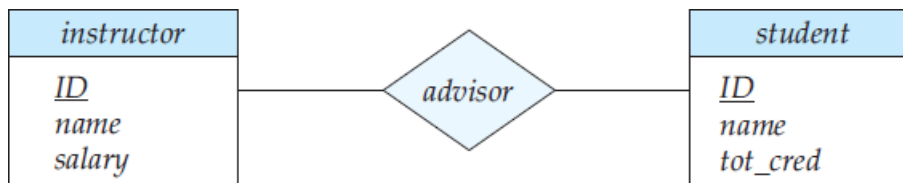


Figure 1: E-R Diagram corresponding to Instructors and Students.

- Consider the E-R diagram in Figure 1, which consists of two entity sets, *instructor* and *student* related through a binary relationship set *advisor*.
- The attributes associated with *instructor* are *ID*, *name*, and *salary*.
- The attributes associated with *student* are *ID*, *name*, and *tot cred*. In Figure 1, attributes of an entity set that are members of the primary key are underlined.
- If a relationship set has some attributes associated with it, then we enclose the attributes in a rectangle and link the rectangle with a dashed line to the diamond representing that relationship set.

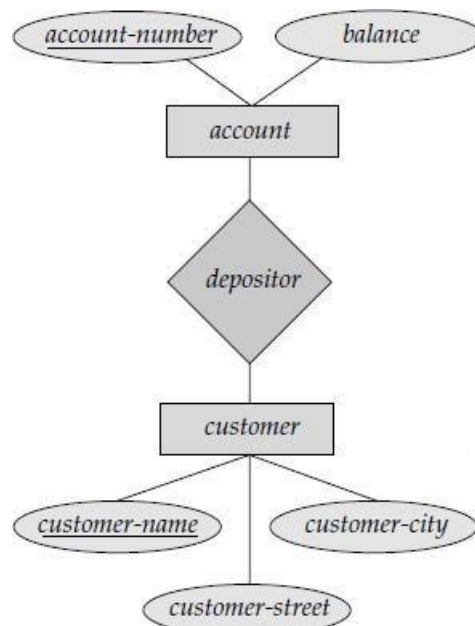


Figure 2: An ER Diagram for Banking Enterprise

- Entity sets are represented by a rectangular box with the entity set name in the header and the attributes listed below it.
- Relationship sets are represented by a diamond connecting a pair of related entity sets.
- The name of the relationship is placed inside the diamond.

- The above E-R diagram indicates that there are two entity sets, *instructor* and *department*, with attributes as outlined earlier.
- The diagram also shows a relationship *member* between *instructor* and *department*.
- In addition to entities and relationships, the E-R model represents certain constraints to which the contents of a database must conform.
- One important constraint is **mapping cardinalities**, which express the number of entities to which another entity can be associated via a relationship set.

Enhanced-ER Model


Discuss enhanced ER model in detail.

- **Enhanced entity-relationship** models, also known as extended **entity-relationship models**, are advanced database **diagrams** very similar to regular **ER diagrams**.
- **Enhanced** ERDs are high-level **models** that represent the requirements and complexities of complex databases.
- It is a diagrammatic technique for displaying the Sub Class and Super Class; Specialization and Generalization; Union or Category; Aggregation etc.

Features of EER Model

- EER creates a design more accurate to database schemas.
- It reflects the data properties and constraints more precisely.
- It includes all modeling concepts of the ER model.
- Diagrammatic technique helps for displaying the EER schema.
- It includes the concept of specialization and generalization.
- It is used to represent a collection of objects that is union of objects of different of different entity types.

A. Sub Class and Super Class

- Sub class and Super class relationship leads the concept of Inheritance.
- The relationship between sub class and super class is denoted with  symbol.

1. Super Class

- Super class is an entity type that has a relationship with one or more subtypes.
- An entity cannot exist in database merely by being member of any super class.
- **For example:** Shape super class is having sub groups as Square, Circle, Triangle.

2. Sub Class

- Sub class is a group of entities with unique attributes.
- Sub class inherits properties and attributes from its super class.
- **For example:** Square, Circle, Triangle are the sub class of Shape super class.

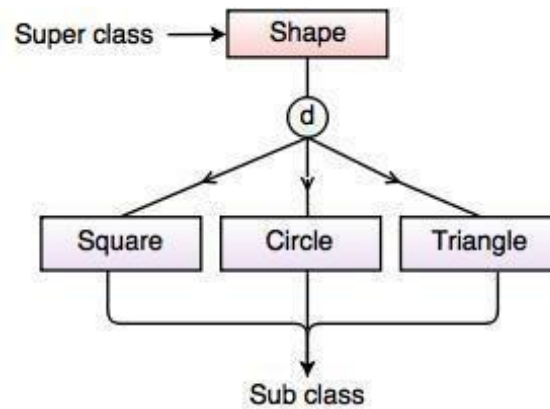


Fig. Super class/Sub class Relationship

B. Specialization and Generalization

1. Generalization

- Generalization is the process of generalizing the entities which contain the properties of all the generalized entities.
- It is a bottom approach, in which two lower level entities combine to form a higher level entity.
- Generalization is the reverse process of Specialization.
- It defines a general entity type from a set of specialized entity type.
- It minimizes the difference between the entities by identifying the common features.

For example:

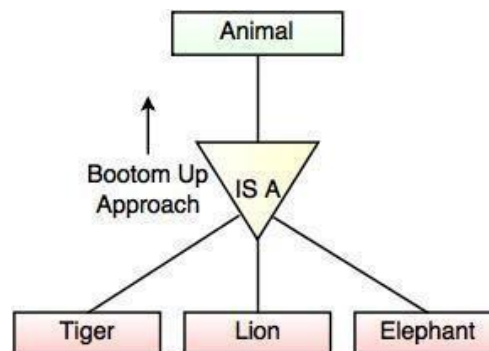


Fig. Generalization

- In the above example, Tiger, Lion, Elephant can all be generalized as Animals.

2. Specialization

- Specialization is a process that defines a group entity which is divided into sub groups based on their characteristic.
- It is a top down approach, in which one higher entity can be broken down into two lower level entities.
- It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member.
- It defines one or more sub class for the super class and also forms the superclass/subclass relationship.

For example

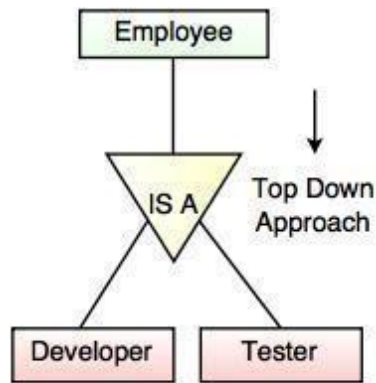


Fig. Specialization

In the above example, Employee can be specialized as Developer or Tester, based on what role they play in an Organization.

C. Category or Union

- Category represents a single super class or sub class relationship with more than one super class.
- It can be a total or partial participation.
- **For example** Car booking, Car owner can be a person, a bank (holds a possession on a Car) or a company.
- Category (sub class) → Owner is a subset of the union of the three super classes → Company, Bank and Person.
- A Category member must exist in at least one of its super classes.

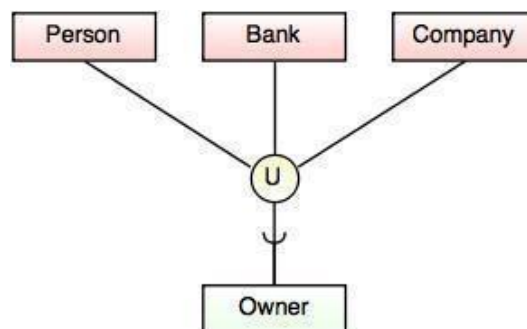


Fig. Categories (Union Type)

D. Aggregation

- Aggregation is a process that represents a relationship between a whole object and its component parts.
- It abstracts a relationship between objects and viewing the relationship as an object.
- It is a process when two entities are treated as a single entity.

II Year / IV Semester - CSE

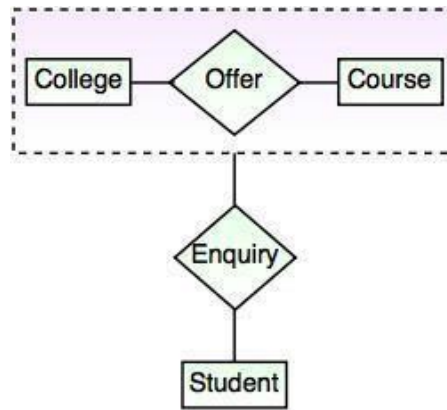


Fig. Aggregation

- In the above example, the relation between College and Course is acting as an Entity in Relation with Student.

Disjointness / Overlap Constraint

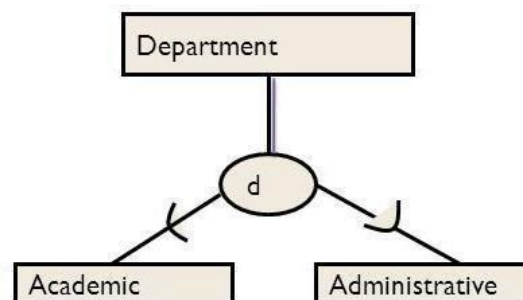
- Specifies that the subclass of the specialization must be disjoint, which means that an entity can be a member of, at most, one subclass of the specialization.
- The d in the specialization circle stands for disjoint.
- If the subclasses are not constrained to be disjoint, they overlap.
- Overlap means that an entity can be a member of more than one subclass of the specialization.
- Overlap constraint is shown by placing an o in the specialization circle.

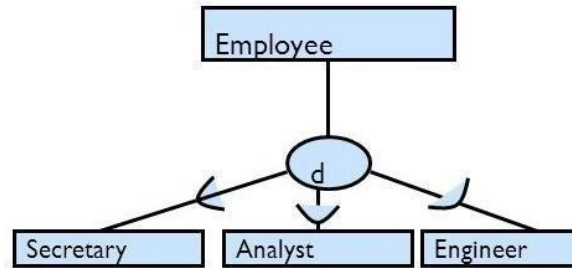
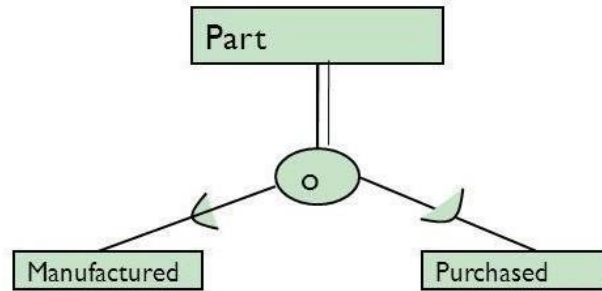
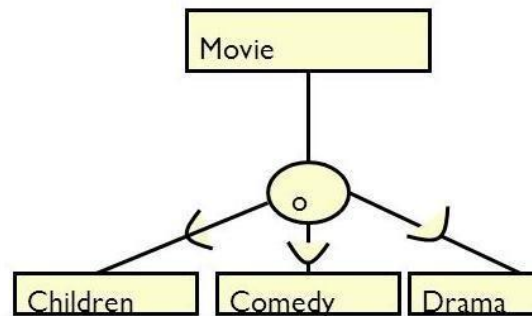
Completeness Constraint

- The completeness constraint may be either total or partial.
- A total specialization constraint specifies that every entity in the superclass must be a member of at least one subclass of the specialization.
- Total specialization is shown by using a double line to connect the super class to the circle.
- A single line is used to display a partial specialization, meaning that an entity does not have to belong to any of the subclasses.

Disjointness vs. Completeness

- Disjoint constraints and completeness constraints are independent. The following possible constraints on specializations are possible:

a) Disjoint, total

b) Disjoint, partial**c) Overlapping, total****d) Overlapping, partial****ER-to-Relational Mapping****Describe ER to Relational mapping in detail.**

- ER diagrams can be mapped to relational schema, that is, it is possible to create relational schema using ER diagram.
- We cannot import all the ER constraints into relational model, but an approximate schema can be generated.
- There are several processes and algorithms available to convert ER Diagrams into Relational Schema. Some of them are automated and some of them are manual.

ER-to-Relational Mapping Algorithm

- Let us use the COMPANY database example to illustrate the mapping procedure.

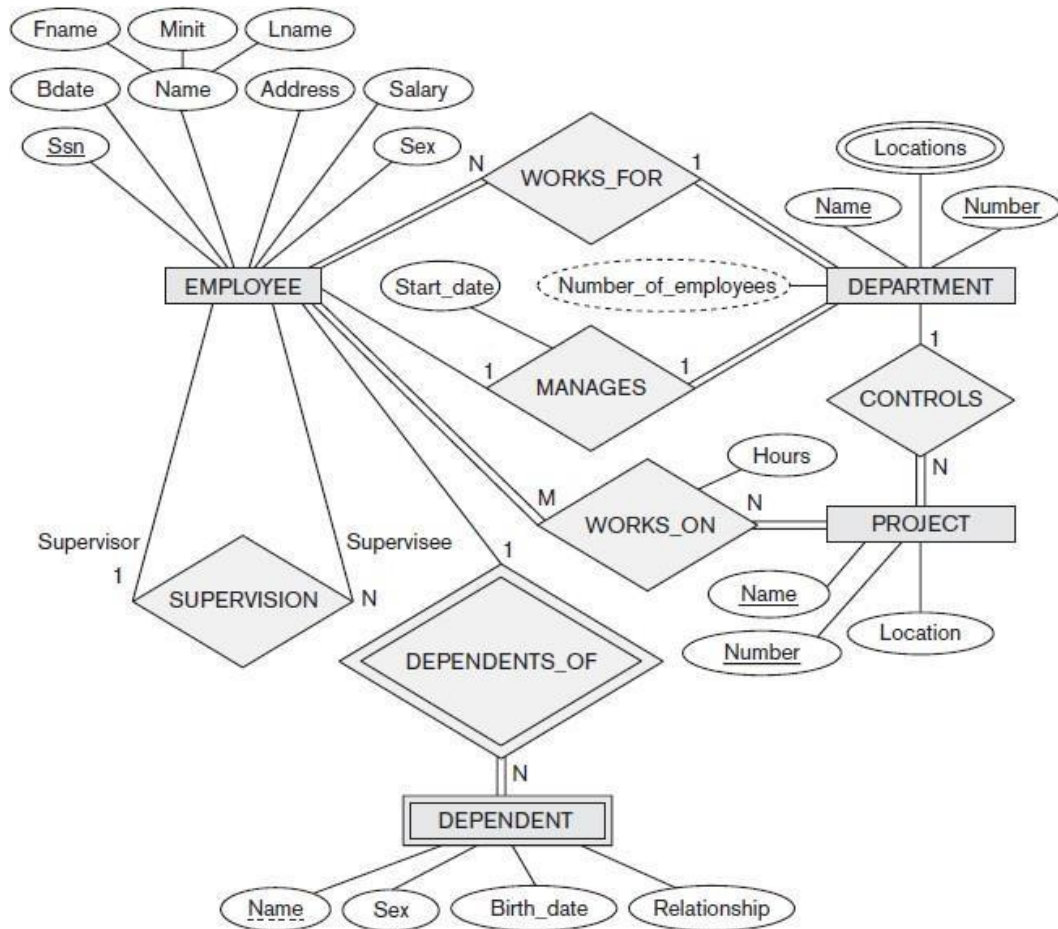


Figure 1: ER conceptual schema diagram for the COMPANY database

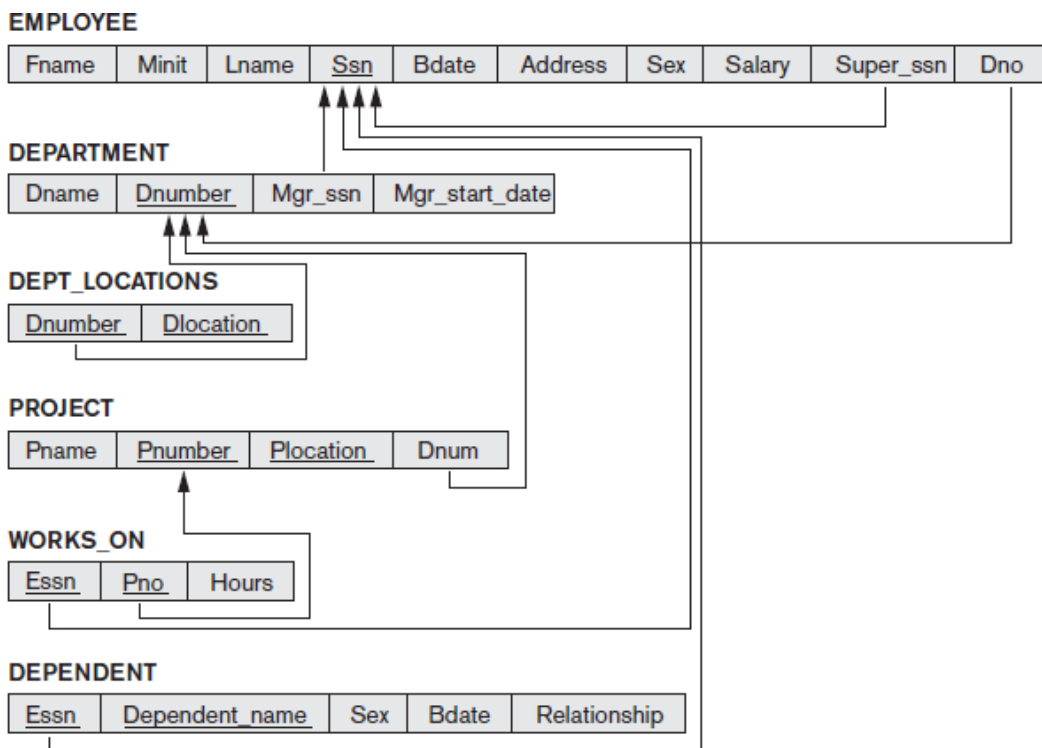


Figure 2: Result of mapping the COMPANY ER schema into a relational database schema

- The COMPANY ER schema is shown again in Figure 1, and the corresponding COMPANY relational database schema is shown in Figure 2 to illustrate the mapping steps.

Step 1: Mapping of Regular Entity Types

- For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E .
- Include only the simple component attributes of a composite attribute. Choose one of the key attributes of E as the primary key for R .
- If the chosen key of E is a composite, then the set of simple attributes that form it will together form the primary key of R .
- If multiple keys were identified for E during the conceptual design, the information describing the attributes that form each additional key is kept in order to specify secondary (unique) keys of relation R .

Step 2: Mapping of Weak Entity Types

- For each weak entity type W in the ER schema with owner entity type E , create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R .
- In addition, include as foreign key attributes of R , the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s); this takes care of mapping the identifying relationship type of W .
- The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W , if any.
- If there is a weak entity type $E2$ whose owner is also a weak entity type $E1$, then $E1$ should be mapped before $E2$ to determine its primary key first.

Step 3: Mapping of Binary 1:1 Relationship Types

- For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R .
- There are three possible approaches:
 - ✓ The foreign key approach
 - ✓ The merged relationship approach
 - ✓ The cross-reference or relationship relation approach

Foreign Key Approach:

- Choose one of the relations— S , say—and include as a foreign key in S the primary key of T .
- It is better to choose an entity type with *total participation* in R in the role of S .
- Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type R as attributes of S .

Merged Relation Approach:

- An alternative mapping of a 1:1 relationship type is to merge the two entity types and the relationship into a single relation.
- This is possible when *both participations are total*, as this would indicate that the two tables will have the exact same number of tuples at all times.

Cross-reference or relationship relation approach:

- The third option is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types.
- As we will see, this approach is required for binary M:N relationships.
- The relation R is called a **relationship relation** (or sometimes a **lookup table**), because each tuple in R represents a relationship instance that relates one tuple from S with one tuple from T .
- The relation R will include the primary key attributes of S and T as foreign keys to S and T .
- The primary key of R will be one of the two foreign keys, and the other foreign key will be a unique key of R .

Step 4: Mapping of Binary 1:N Relationship Types

- For each regular binary 1:N relationship type R , identify the relation S that represents the participating entity type at the N -side of the relationship type.
- Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R ; we do this because each entity instance on the N -side is related to at most one entity instance on the 1 -side of the relationship type.
- Include any simple attributes (or simple components of composite attributes) of the 1:N relationship type as attributes of S .

Step 5: Mapping of Binary M:N Relationship Types

- For each binary M:N relationship type R , create a new relation S to represent R .
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their *combination* will form the primary key of S . Also include any simple attributes of the M:N relationship type (or simple components of composite attributes) as attributes of S .
- Notice that we cannot represent an M:N relationship type by a single foreign key attribute in one of the participating relations (as we did for 1:1 or 1:N relationship types) because of the M:N cardinality ratio; we must create a separate *relationship relation* S .

Step 6: Mapping of Multi-valued Attributes

- For each multi-valued attribute A , create a new relation R .
- This relation R will include an attribute corresponding to A , plus the primary key attribute K —as a foreign key in R —of the relation that represents the entity type or relationship type that has A as a multi-valued attribute.
- The primary key of R is the combination of A and K . If the multi-valued attribute is composite, we include its simple components.

Step 7: Mapping of N -ary Relationship Types

- For each n -ary relationship type R , where $n > 2$, create a new relation S to represent R .
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.

- Also include any simple attributes of the n -ary relationship type (or Step 7: Mapping of N -ary Relationship Types)

Step 8: Options for Mapping Specialization or Generalization

- Convert each specialization with m subclasses $\{S_1, S_2, \dots, S_m\}$ and (generalized) superclass C , where the attributes of C are $\{k, a_1, \dots, a_n\}$ and k is the (primary) key, into relation schemas using one of the following options:
 - **Option 8A: Multiple relations—superclass and subclasses**
 - ✓ Create a relation L for C with attributes $\text{Attrs}(L) = \{k, a_1, \dots, a_n\}$ and $\text{PK}(L) = k$. Create a relation L_i for each subclass S_i , $1 \leq i \leq m$, with the attributes $\text{Attrs}(L_i) = \{k\} \cup \{\text{attributes of } S_i\}$ and $\text{PK}(L_i) = k$.
 - ✓ This option works for any specialization (total or partial, disjoint or overlapping).
 - **Option 8B: Multiple relations—subclass relations only**
 - ✓ Create a relation L_i for each subclass S_i , $1 \leq i \leq m$, with the attributes $\text{Attrs}(L_i) = \{\text{attributes of } S_i\} \cup \{k, a_1, \dots, a_n\}$ and $\text{PK}(L_i) = k$.
 - ✓ This option only works for a specialization whose subclasses are *total* (every entity in the superclass must belong to (at least) one of the subclasses).
 - ✓ Additionally, it is only recommended if the specialization has the *disjointness constraint*. If the specialization is *overlapping*, the same entity may be duplicated in several relations.
 - **Option 8C: Single relation with one type attribute.**
 - ✓ Create a single relation L with attributes $\text{Attrs}(L) = \{k, a_1, \dots, a_n\} \cup \{\text{attributes of } S_1\} \cup \dots \cup \{\text{attributes of } S_m\} \cup \{t\}$ and $\text{PK}(L) = k$.
 - ✓ The attribute t is called a **type** (or **discriminating**) attribute whose value indicates the subclass to which each tuple belongs, if any.
 - ✓ This option works only for a specialization whose subclasses are *disjoint*, and has the potential for generating many NULL values if many specific attributes exist in the subclasses.
 - **Option 8D: Single relation with multiple type attributes.**
 - ✓ Create a single relation schema L with attributes $\text{Attrs}(L) = \{k, a_1, \dots, a_n\} \cup \{\text{attributes of } S_1\} \cup \dots \cup \{\text{attributes of } S_m\} \cup \{t_1, t_2, \dots, t_m\}$ and $\text{PK}(L) = k$.
 - ✓ Each t_i , $1 \leq i \leq m$, is a **Boolean type attribute** indicating whether a tuple belongs to subclass S_i .
 - ✓ This option is used for a specialization whose subclasses are *overlapping* (but will also work for a disjoint specialization).

Step 9: Mapping of Shared Subclasses (Multiple Inheritance)

- A shared subclass is a subclass of several superclasses, indicating multiple inheritance.
- These classes must all have the same key attribute; otherwise, the shared subclass would be modeled as a category (union type) as we discussed in Section 8.4.
- We can apply any of the options discussed in step 8 to a shared subclass, subject to the restrictions discussed in step 8 of the mapping algorithm.

Step 10: Mapping of Union Types (Categories)

- For mapping a category whose defining superclasses have different keys, it is customary to specify a new key attribute, called a **surrogate key**, when creating a relation to correspond to the category.
- The keys of the defining classes are different, so we cannot use any one of them exclusively to identify all entities in the category.

Correspondence between ER and Relational Models

ER MODEL	RELATIONAL MODEL
Entity type	Entity relation
1:1 or 1:N relationship type	Foreign key (or <i>relationship</i> relation)
M:N relationship type	<i>Relationship</i> relation and <i>two</i> foreign keys
<i>n</i> -ary relationship type	<i>Relationship</i> relation and <i>n</i> foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

Functional Dependencies**What is Functional Dependency? (Or) Explain the concept of functional dependency in detail.**

- **Functional Dependency** is a relationship that exists between multiple attributes of a relation.
- This concept is given by **E. F. Codd**.
- Functional dependency represents a formalism on the infrastructure of relation.
- It is a type of constraint existing between various attributes of a relation.
- It is used to define various normal forms.
- These dependencies are restrictions imposed on the data in database.
- If P is a relation with A and B attributes, a functional dependency between these two attributes is represented as $\{A \rightarrow B\}$.
- It specifies that,

A	It is a determinant set.
B	It is a dependent attribute.
$\{A \rightarrow B\}$	A functionally determines B. B is a functionally dependent on A.

- Each value of A is associated precisely with one B value. A functional dependency is trivial if B is a subset of A.
- 'A' Functionality determines 'B' $\{A \rightarrow B\}$ (Left hand side attributes determine the values of Right hand side attributes).

(Or)

- Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A_1, A_2, \dots, A_n , then those two tuples must have to have same values for attributes B_1, B_2, \dots, B_n .
- Functional dependency is represented by an arrow sign (\rightarrow) that is, $X \rightarrow Y$, where X functionally determines Y . The left-hand side attributes determine the values of attributes on the right-hand side.

Design goals

- Avoid redundant data.
- Ensure that relationships among attributes are represented.
- Facilitate the checking of updates for violation of database integrity constraints.

For example: <Employee> Table

EmpId	EmpName

- In the above <Employee> table, EmpName (employee name) is functionally dependent on EmpId (employee id) because the EmpId is unique for individual names.
- The EmpId identifies the employee specifically, but EmpName cannot distinguish the EmpId because more than one employee could have the same name.
- The functional dependency between attributes eliminates the repetition of information.
- It is related to a candidate key, which uniquely identifies a tuple and determines the value of all other attributes in the relation.

Types of Dependency

• Full Functional Dependencies

- ✓ In a relation R , X and Y are attributes. X functionally determines Y ($X \rightarrow Y$). subset of X should not functionally determine Y

$Student_no \rightarrow Marks \leftarrow course_no$

In the above example marks is fully functionally dependent on student_no and course_no together and not on subset of student_no, course_no.

• Partial Dependencies

- ✓ Attribute Y is partially dependent on attribute X only if it is dependent on a subset of attribute X .

For example course_name, Instructor_name are partially dependent on composite attributes (student_no, course_no} because course_no only defines course_name, Instructor_name.

• The main characteristics of functional dependencies

- ✓ Have a one-to-one relationship between attributes on the left-and right-hand side of a dependency is nontrivial.
- ✓ A dependency is trivial if and only if, the right-hand side is a subset of the left side.

Properties of Functional Dependencies (Or) Inference rules for Functional Dependencies

Armstrong's Axioms

- Armstrong's Axioms is a set of rules.
- It provides a simple technique for reasoning about functional dependencies.
- It was developed by William W. Armstrong in 1974.
- It is used to infer all the functional dependencies on a relational database.
- If F is a set of functional dependencies then the closure of F, denoted as F^+ , is the set of all functional dependencies logically implied by F.
- Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

Various Axioms Rules**A. Primary Rules**

Rule 1	Reflexivity Rule If A is a set of attributes and B is a subset of A, then A holds B. $\{ A \rightarrow B \}$
Rule 2	Augmentation Rule If A hold B and C is a set of attributes, then AC holds BC. $\{ AC \rightarrow BC \}$ It means that attribute in dependencies does not change the basic dependencies.
Rule 3	Transitivity Rule If A holds B and B holds C, then A holds C. If $\{ A \rightarrow B \}$ and $\{ B \rightarrow C \}$, then $\{ A \rightarrow C \}$ A holds B $\{ A \rightarrow B \}$ means that A functionally determines B.

B. Secondary Rules

Rule 1	Union If A holds B and A holds C, then A holds BC. If $\{ A \rightarrow B \}$ and $\{ A \rightarrow C \}$, then $\{ A \rightarrow BC \}$
Rule 2	Decomposition If A holds BC and A holds B, then A holds C. If $\{ A \rightarrow BC \}$ and $\{ A \rightarrow B \}$, then $\{ A \rightarrow C \}$
Rule 3	Pseudo Transitivity If A holds B and BC holds D, then AC holds D. If $\{ A \rightarrow B \}$ and $\{ BC \rightarrow D \}$, then $\{ AC \rightarrow D \}$

Sometimes Functional Dependency Sets are not able to reduce if the set has following properties,

- The Right-hand side set of functional dependency holds only one attribute.
- The Left-hand side set of functional dependency cannot be reduced, it changes the entire content of the set.
- Reducing any functional dependency may change the content of the set.
- A set of functional dependencies with the above three properties are also called as **Canonical or Minimal**.

Trivial Functional Dependency

Trivial	If A holds B $\{A \rightarrow B\}$, where B is a subset of A, then it is called a Trivial Functional Dependency . Trivial always holds Functional Dependency.
Non-Trivial	If A holds B $\{A \rightarrow B\}$, where B is not a subset A, then it is called as a Non-Trivial Functional Dependency .
Completely Non-Trivial	If A holds B $\{A \rightarrow B\}$, where $A \cap Y = \Phi$, it is called as a Completely Non-Trivial Functional Dependency .

Advantages of Functional Dependency

- Functional Dependency avoids data redundancy where same data should not be repeated at multiple locations in same database.
- It maintains the quality of data in database.
- It allows clearly defined meanings and constraints of databases.
- It helps in identifying bad designs.
- It expresses the facts about the database design.

Example 1:

Consider relation E = (P, Q, R, S, T, U) having set of Functional Dependencies (FD).

$$\begin{array}{ll}
 P \rightarrow Q & P \rightarrow R \\
 QR \rightarrow S & Q \rightarrow T \\
 QR \rightarrow U & PR \rightarrow U
 \end{array}$$

Calculate some members of Axioms are as follows,

1. $P \rightarrow T$
2. $PR \rightarrow S$
3. $QR \rightarrow SU$
4. $PR \rightarrow SU$

Solution:**1. $P \rightarrow T$**

In the above FD set, $P \rightarrow Q$ and $Q \rightarrow T$

So, Using Transitive Rule: If $\{A \rightarrow B\}$ and $\{B \rightarrow C\}$, then $\{A \rightarrow C\}$

\therefore If $P \rightarrow Q$ and $Q \rightarrow T$, then **$P \rightarrow T$** .

 $P \rightarrow T$ **2. $PR \rightarrow S$**

In the above FD set, $P \rightarrow Q$

As, $QR \rightarrow S$

So, Using Pseudo Transitivity Rule: If $\{A \rightarrow B\}$ and $\{BC \rightarrow D\}$, then $\{AC \rightarrow D\}$

\therefore If $P \rightarrow Q$ and $QR \rightarrow S$, then **$PR \rightarrow S$** .

 $PR \rightarrow S$

3. QR → SU

In above FD set, $QR \rightarrow S$ and $QR \rightarrow U$

So, Using Union Rule: If $\{A \rightarrow B\}$ and $\{A \rightarrow C\}$, then $\{A \rightarrow BC\}$

∴ If $QR \rightarrow S$ and $QR \rightarrow U$, then **$QR \rightarrow SU$** .

QR → SU**4. PR → SU**

So, Using Pseudo Transitivity Rule: If $\{A \rightarrow B\}$ and $\{BC \rightarrow D\}$, then $\{AC \rightarrow D\}$

∴ If $PR \rightarrow S$ and $PR \rightarrow U$, then **$PR \rightarrow SU$** .

PR → SU**Example 2:**

Let us consider the example of schema $R = (A, B, C, G, H, I)$ and the set F of functional dependencies $\{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$.

We list several members of F^+ here:

- $A \rightarrow H$. Since $A \rightarrow B$ and $B \rightarrow H$ hold, we apply the transitivity rule. Observe that it was much easier to use Armstrong's axioms to show that $A \rightarrow H$ holds than it was to argue directly from the definitions, as we did earlier in this section.
- $CG \rightarrow HI$. Since $CG \rightarrow H$ and $CG \rightarrow I$, the union rule implies that $CG \rightarrow HI$.
- $AG \rightarrow I$. Since $A \rightarrow C$ and $CG \rightarrow I$, the pseudotransitivity rule implies that $AG \rightarrow I$ holds.
- Another way of finding that $AG \rightarrow I$ holds is as follows: We use the augmentation rule on $A \rightarrow C$ to infer $AG \rightarrow CG$. Applying the transitivity rule to this dependency and $CG \rightarrow I$, we infer $AG \rightarrow I$.

Non-loss Decomposition

Explain in detail about non-loss decomposition and functional dependencies.

What is Decomposition?

- Decomposition is the process of breaking down in parts or elements.
- It replaces a relation with a collection of smaller relations.
- It breaks the table into multiple tables in a database.
- It should always be lossless, because it confirms that the information in the original relation can be accurately reconstructed based on the decomposed relations.
- If there is no proper decomposition of the relation, then it may lead to problems like loss of information.

Properties of Decomposition

The following are the properties of decomposition,

1. Lossless Decomposition
2. Dependency Preservation
3. Lack of Data Redundancy

1. Lossless Decomposition

- Decomposition must be lossless. It means that the information should not get lost from the relation that is decomposed.
- It gives a guarantee that the join will result in the same relation as it was decomposed.

Example:

- Let's take 'E' is the Relational Schema, With instance 'e'; is decomposed into: E1, E2, E3, En; With instance: e1, e2, e3, en, If $e1 \bowtie e2 \bowtie e3 \dots \bowtie en$, then it is called as '**Lossless Join Decomposition**'.
- In the above example, it means that, if natural joins of all the decomposition give the original relation, then it is said to be lossless join decomposition.

Example: <Employee_Department> Table

Eid	Ename	Age	City	Salary	Deptid	DeptName
E001	ABC	29	Pune	20000	D001	Finance
E002	PQR	30	Pune	30000	D002	Production
E003	LMN	25	Mumbai	5000	D003	Sales
E004	XYZ	24	Mumbai	4000	D004	Marketing
E005	STU	32	Bangalore	25000	D005	Human Resource

- Decompose the above relation into two relations to check whether decomposition is lossless or lossy.
- Now, we have decomposed the relation that is Employee and Department.

Relation 1: <Employee> Table

Eid	Ename	Age	City	Salary
E001	ABC	29	Pune	20000
E002	PQR	30	Pune	30000
E003	LMN	25	Mumbai	5000
E004	XYZ	24	Mumbai	4000
E005	STU	32	Bangalore	25000

Employee Schema contains (Eid, Ename, Age, City, Salary).

Relation 2: <Department> Table

Deptid	Eid	DeptName
D001	E001	Finance
D002	E002	Production
D003	E003	Sales
D004	E004	Marketing
D005	E005	Human Resource

- Department Schema contains (Deptid, Eid, DeptName).
- So, the above decomposition is a Lossless Join Decomposition, because the two relations contains one common field that is 'Eid' and therefore join is possible.
- Now apply natural join on the decomposed relations.

Employee \bowtie Department

Eid	Ename	Age	City	Salary	Deptid	DeptName
E001	ABC	29	Pune	20000	D001	Finance
E002	PQR	30	Pune	30000	D002	Production
E003	LMN	25	Mumbai	5000	D003	Sales
E004	XYZ	24	Mumbai	4000	D004	Marketing

E005	STU	32	Bangalore	25000	D005	Human Resource
------	-----	----	-----------	-------	------	----------------

Hence, the decomposition is Lossless Join Decomposition.

- If the <Employee> table contains (Eid, Ename, Age, City, Salary) and <Department> table contains (Deptid and DeptName), then it is not possible to join the two tables or relations, because there is no common column between them. And it becomes **Lossy Join Decomposition**.

2. Dependency Preservation

- Dependency is an important constraint on the database.
- Every dependency must be satisfied by at least one decomposed table.
- If $\{A \rightarrow B\}$ holds, then two sets are functional dependent. And, it becomes more useful for checking the dependency easily if both sets in a same relation.
- This decomposition property can only be done by maintaining the functional dependency.
- In this property, it allows to check the updates without computing the natural join of the database structure.

3. Lack of Data Redundancy

- Lack of Data Redundancy is also known as a **Repetition of Information**.
- The proper decomposition should not suffer from any data redundancy.
- The careless decomposition may cause a problem with the data.
- The lack of data redundancy property may be achieved by Normalization process.

Normalization

Explain in detail about normalization. (Or) What are Normal Forms? Explain the types of normal forms with an example. (Nov/Dec 2014) (Or) State the need for normalization of a database and explain the various forms with suitable examples. (April/May 2015) (Or) Explain first normal form, second normal form, third normal form and BCNF with an example. (Nov/Dec 2016) (Or) What is database Normalization? Explain first normal form, second normal form, third normal form with an example. (April/May 2018) (Or) Give an example of a relation that is in 3NF but not in BCNF. How will you convert that relation into BCNF. (Nov/Dec 2018)

- Normalization** is a process of organizing the data in the database.
- It is a systematic approach used to minimize the redundancy from a relation or set of relations.
- It is used to avoid / eliminate data redundancy, insertion anomaly, update anomaly & deletion anomaly.
- It was developed by **E. F. Codd**.

(Or)

- —Normalization is a process of designing a consistent database by minimizing redundancy and ensuring data integrity through decomposition which is lossless.||
- The goal is to generate a set of relation schemas that allows us to store information without unnecessary redundancy, yet also allows us to retrieve information easily.
- The approach is to design schemas that are in an appropriate *normal form*.

- To determine whether a relation schema is in one of the desirable normal forms, we need additional information about the real-world enterprise that we are modeling with the database.
- The most common approach is to use **functional dependencies**.
- It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.
- It is also called as Canonical Synthesis.
- Normalization is used for mainly two purposes,
 - ✓ Eliminating redundant (useless) data.
 - ✓ Ensuring data dependencies make sense i.e data is logically stored.

Features of Normalization

- Normalization avoids the data redundancy.
- It is a formal process of developing data structures.
- It promotes the data integrity.
- It ensures data dependencies make sense that means data is logically stored.
- It eliminates the undesirable characteristics like Insertion, Update and Deletion Anomalies.

Anomalies in DBMS

- There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly.

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

Update Anomaly:

- In the above table we have two rows for employee Rick as he belongs to two departments of the company.
- If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent.
- If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert Anomaly:

- Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete Anomaly:

- Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

First, Second, Third Normal Forms**Types of Normalization**

To overcome these anomalies we need to normalize the data.

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transitive dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

(Or)

Following are the types of Normalization:

- First Normal Form
- Second Normal Form
- Third Normal Form
- Fourth Normal Form
- Fifth Normal Form
- BCNF (Boyce – Codd Normal Form)
- DKNF (Domain Key Normal Form)

1. First Normal Form (1NF)

- First Normal Form (1NF) is a simple form of Normalization.
- It simplifies each attribute in a relation.
- In 1NF, there should not be any repeating group of data.
- Each set of column must have a unique value.
- It contains atomic values because the table cannot hold multiple values.

Rules of First Normal Form

- For a table to be in the First Normal Form, it should follow the following 4 rules:
 - ✓ It should only have single (atomic) valued attributes/columns.
 - ✓ Values stored in a column should be of the same domain
 - ✓ All the columns in a table should have unique names.
 - ✓ And the order in which data is stored, does not matter.

Example: Employee Table

ECode	Employee_Name	Department_Name
-------	---------------	-----------------

II Year / IV Semester - CSE

1	ABC	Sales, Production
2	PQR	Human Resource
3	XYZ	Quality Assurance, Marketing

Employee Table using 1NF

ECode	Employee_Name	Department_Name
1	ABC	Sales
1	ABC	Production
2	PQR	Human Resource
3	XYZ	Quality Assurance
3	XYZ	Marketing

2. Second Normal Form (2NF)

- In 2NF, the table is required in 1NF.
- The main rule of 2NF is, 'No non-prime attribute is dependent on the proper subset of any candidate key of the table.'
- An attribute which is not part of candidate key is known as non-prime attribute.

Example: Employee Table using 1NF

ECode	Employee_Name	Employee_Age
1	ABC	38
1	ABC	38
2	PQR	38
3	XYZ	40
3	XYZ	40

Candidate Key: ECode, Employee_Name

Non prime Attribute: Employee_Age

- The above table is in 1NF. Each attribute has atomic values.
- However, it is not in 2NF because non prime attribute Employee_Age is dependent on ECode alone, which is a proper subset of candidate key.
- This violates the rule for 2NF as the rule says 'No non-prime attribute is dependent on the proper subset of any candidate key of the table'.

2NF (Second Normal Form):**Employee1 Table**

ECode	Employee_Age
1	38
2	38
3	40

Employee 2 Table

ECode	Employee_Name
1	ABC
1	ABC
2	PQR
3	XYZ
3	XYZ

- Now, the above tables comply with the Second Normal Form (2NF).

3. Third Normal Form (3NF)

- Third Normal Form (3NF) is used to minimize the transitive redundancy.
- In 3NF, the table is required in 2NF.
- While using the 2NF table, there should not be any transitive partial dependency.
- 3NF reduces the duplication of data and also achieves the data integrity.

(Or)

- A table is said to be in the Third Normal Form when,
 - ✓ It is in the Second Normal form.
 - ✓ And, it doesn't have Transitive Dependency.

Example : <Employee> Table

EId	Ename	DOB	City	State	Zip
001	ABC	10/05/1990	Pune	Maharashtra	411038
002	XYZ	11/05/1988	Mumbai	Maharashtra	400007

- In the above <Employee> table, EId is a primary key but City, State depends upon Zip code.
- The dependency between Zip and other fields is called Transitive Dependency.
- Therefore we apply 3NF. So, we need to move the city and state to the new <Employee_Table2> table, with Zip as a Primary key.

<Employee_Table1> Table

EId	Ename	DOB	Zip
001	ABC	10/05/1990	411038
002	XYZ	11/05/1988	400007

<Employee_Table2> Table

City	State	Zip
Pune	Maharashtra	411038
Mumbai	Maharashtra	400007

- The advantage of removing transitive dependency is, it reduces the amount of data dependencies and achieves the data integrity.
- In the above example, using with the 3NF, there is no redundancy of data while inserting the new records.

- The City, State and Zip code will be stored in the separate table. And therefore the updation becomes more easier because of no data redundancy.

4. BCNF (Boyce – Code Normal Form)

- BCNF which stands for Boyce – Code Normal Form is developed by Raymond F. Boyce and E. F. Codd in 1974.
- BCNF is a higher version of 3NF.
- It deals with the certain type of anomaly which is not handled by 3NF.
- A table complies with BCNF if it is in 3NF and any attribute is fully functionally dependent that is $A \rightarrow B$. (Attribute 'A' is determinant).
- If every determinant is a candidate key, then it is said to be BCNF.
- Candidate key has the ability to become a primary key. It is a column in a table.

(Or)

- Boyce and Codd Normal Form** is a higher version of the Third Normal form.
- This form deals with certain type of anomaly that is not handled by 3NF.
- A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.
- For a table to be in BCNF, following conditions must be satisfied:
 - ✓ R must be in 3rd Normal Form
 - ✓ and, for each functional dependency ($X \rightarrow Y$), X should be a super Key.

Example : <EmployeeMain> Table

Empid	Ename	DeptName	DeptType
E001	ABC	Production	D001
E002	XYZ	Sales	D002

The functional dependencies are:

$Empid \rightarrow EmpName$

$DeptName \rightarrow DeptType$

Candidate Key:

Empid

DeptName

- The above table is not in BCNF as neither Empid nor DeptName alone are keys.
- We can break the table in three tables to make it comply with BCNF.

<Employee> Table

Empid	EmpName
E001	ABC
E002	XYZ

<Department> Table

DeptName	DeptType
Production	D001
Sales	D002

<Emp_Dept> Table

Empid	DeptName
E001	Production
E002	Sales

Now, the functional dependencies are:

Empid → EmpName

DeptName → DeptType

Candidate Key:

<Employee> Table : Empid

<Department> Table : DeptType

<Emp_Dept> Table : Empid, DeptType

- So, now both the functional dependencies left side part is a key, so it is in the BCNF.

5. Fourth Normal Form (4NF)

- Fourth Normal Form (4NF) does not have non-trivial multi-valued dependencies other than a candidate key.
- 4NF builds on the first three normal forms (1NF, 2NF and 3NF) and the BCNF.
- It does not contain more than one multi-valued dependency.
- This normal form is rarely used outside of academic circles.

Rules for 4th Normal Form

For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

- It should be in the **Boyce-Codd Normal Form**.
- And, the table should not have any **Multi-valued Dependency**.

For Example:

- A table contains a list of three things that is 'Student', 'Teacher', 'Book'. Teacher is in charge of Student and recommended book for each student.
- These three elements (Student, Teacher and Book) are independent of oneanother.
- Changing the student's recommended book, for instance, has no effect on the student itself. This is an example of multi-valued dependency, where an item depends on more than one value. In this example, the student depends on both teacher and book.
- Therefore, 4NF states that a table should not have more than one dependency.

6. Fifth Normal Form (5NF)

- 5NF is also known as Project-Join Normal Form (PJ/NF).
- It is designed for reducing the redundancy in relational databases.
- 5NF requires semantically related multiple relationships, which are rare.
- In 5NF, if an attribute is multivalued attribute, then it must be taken out as a separate entity.
- While performing 5NF, the table must be in 4NF.

7. DKNF (Domain Key Normal Form)

- DKNF stands for Domain Key Normal Form requires the database that contains no constraints other than domain constraints and key constraints.
- In DKNF, it is easy to build a database.
- It avoids general constraints in the database which are not clear domain or key constraints.
- The 3NF, 4NF, 5NF and BCNF are special cases of the DKNF.
- It is achieved when every constraint on the relation is a logical consequence of the definition.

Dependency Preservation**Explain in detail about dependency preservation.**

- F' is a set of functional **dependencies** on schema R , but in general, However, it may be that .
- If this is so, then every functional **dependency** in F is implied by F' , and if F' is satisfied, then F must also be satisfied.
- A decomposition having the property that is a **dependency-preserving** decomposition.

Dependency Preservation

- A Decomposition $D = \{ R_1, R_2, R_3, \dots, R_n \}$ of R is dependency preserving with respect to a set F of Functional dependency if

$$(F_1 \cup F_2 \cup \dots \cup F_m)^+ = F^+$$

Consider a relation R

$R \rightarrow F \{ \dots \text{with some functional dependency (FD)} \dots \}$

R is decomposed or divided into R_1 with $FD \{ f_1 \}$ and R_2 with $\{ f_2 \}$, then there can be three cases:

$f_1 \cup f_2 = F \rightarrow$ Decomposition is dependency preserving.

$f_1 \cup f_2$ is a subset of $F \rightarrow$ Not Dependency preserving.

$f_1 \cup f_2$ is a super set of $F \rightarrow$ This case is not possible.

Problem:

Let a relation $R (A, B, C, D)$ and functional dependency $\{ AB \rightarrow C, C \rightarrow D, D \rightarrow A \}$.

Relation R is decomposed into $R_1(A, B, C)$ and $R_2(C, D)$. Check whether decomposition is dependency preserving or not.

Solution:

$R_1(A, B, C)$ and $R_2(C, D)$

Let us find closure of F_1 and F_2

To find closure of F_1 , consider all combination of

ABC . i.e., find closure of A, B, C, AB, BC and AC

Note ABC is not considered as it is always ABC

$\text{closure}(A) = \{ A \}$ // Trivial

$\text{closure}(B) = \{ B \}$ // Trivial

$\text{closure}(C) = \{ C, A, D \}$ but D can't be in closure as D is not present in R_1 .
 $= \{ C, A \}$

$C \rightarrow A$ // Removing C from right side as it is trivial attribute

$$\begin{aligned}\text{closure}(AB) &= \{A, B, C, D\} \\ &= \{A, B, C\}\end{aligned}$$

AB \rightarrow C // Removing AB from right side as these are trivial attributes

$$\begin{aligned}\text{closure}(BC) &= \{B, C, D, A\} \\ &= \{A, B, C\}\end{aligned}$$

BC \rightarrow A // Removing BC from right side as these are trivial attributes

$$\text{closure}(AC) = \{A, C, D\}$$

AC \rightarrow D // Removing AC from right side as these are trivial attributes

F1 {C \rightarrow A, AB \rightarrow C, BC \rightarrow A}.

Similarly F2 { C \rightarrow D }

In the original Relation Dependency { AB \rightarrow C , C \rightarrow D , D \rightarrow A }.

AB \rightarrow C is present in F1.

C \rightarrow D is present in F2.

D \rightarrow A is not preserved.

F1 \cup F2 is a subset of F. So **given decomposition is not dependency preserving.**

Dependency-Preserving Decomposition

- The **dependency preservation decomposition** is another property of decomposed relational database schema D in which each functional dependency X \rightarrow Y specified in F either appeared directly in one of the relation schemas R_i in the decomposed D or could be inferred from the dependencies that appear in some R_i.
- Decomposition D = { R₁ , R₂, R₃,..., ,R_m} of R is said to be dependency-preserving with respect to F if the union of the projections of F on each R_i , in D is equivalent to F.
- In other words, R \subset join of R₁, R₁ over X.
- The dependencies are preserved because each dependency in F represents a constraint on the database.
- If decomposition is not dependency-preserving, some dependency is lost in the decomposition.

Example:

Let a relation R(A,B,C,D) and set a FDs F = { A \rightarrow B , A \rightarrow C , C \rightarrow D } are given.

A relation R is decomposed into -

R₁ = (A, B, C) with FDs F₁ = { A \rightarrow B, A \rightarrow C }, and

R₂ = (C, D) with FDs F₂ = { C \rightarrow D }.

$$F' = F_1 \cup F_2 = \{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$$

$$\text{so, } F' = F.$$

And so, F'⁺ = F⁺.

Thus, the decomposition is dependency preserving decomposition.

Multi-valued Dependencies and Fourth Normal Form**What is Multi-valued Dependency?**

A table is said to have multi-valued dependency, if the following conditions are true,

1. For a dependency $A \twoheadrightarrow B$, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
3. And, for a relation $R(A,B,C)$, if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.

If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

- To deal with the problem of BCNF, R. Fagin introduced the idea of multi-valued dependency (MVD) and the fourth normal form (4NF).
- A multi-valued dependency (MVD) is a functional dependency where the dependency may be to a set and not just a single value.
- It is defined as $X \twoheadrightarrow Y$ in relation $R(X, Y, Z)$, if each X value is associated with a set of Y values in a way that does not depend on the Z values.
- Here X and Y are both subsets of R. The notation $X \twoheadrightarrow Y$ is used to indicate that a set of attributes of Y shows a multi-valued dependency (MVD) on a set of attributes of X.

Join Dependencies and Fifth Normal Form

- The anomalies of MVDs are eliminated by join dependency (JD) and 5NF.
- A join dependency (JD) can be said to exist if the join of R_1 and R_2 over C is equal to relation R.
- Where, R_1 and R_2 are the decompositions $R_1(A, B, C)$, and $R_2(C,D)$ of a given relations $R(A, B, C, D)$. Alternatively, R_1 and R_2 is a lossless decomposition of R.
- In other words, $(A, B, C, D), (C, D)$ will be a join dependency of R if the join of the join's attributes is equal to relation R. Here, (R_1, R_2, R_3, \dots) indicates that relations R_1, R_2, R_3 and so on are a join dependency (JD) of R.
- Therefore, a necessary condition for a relation R to satisfy a JD (R_1, R_2, \dots, R_n) is that R.

Denormalization**What is Denormalization?**

- Denormalization is the process of increasing the redundancy in the database.
- It is the opposite process of normalization.
- It is mostly done for improving the performance.
- It is a strategy that database managers use to increase the performance of a database structure.
- Denormalization adds redundant data normalized database for reducing the problems with database queries which combine data from the various tables into a single table.
- The process of adding redundant data to get rid of complex join, in order to optimize database performance. This is done to speed up database access by moving from higher to lower form of normalization.

- Data is included in one table from another in order to eliminate the second table which reduces the number of JOINS in a query and thus achieves performance.

Difference between Lossless Join Decomposition and Dependency Preservation Decomposition

Lossless Join Decomposition

- The lossless join property is a feature of decomposition supported by normalization. It is the ability to ensure that any instance of the original relation can be identified from corresponding instances in the smaller relations.

R : relation, F : set of functional dependencies on R,

X,Y : decomposition of R

- A decomposition $\{R_1, R_2, \dots, R_n\}$ of a relation R is called a lossless decomposition for R if the natural join of R_1, R_2, \dots, R_n produces exactly the relation R.

- A decomposition is lossless if we can recover:

$R(A, B, C) \rightarrow \text{Decompose} \rightarrow R_1(A, B) R_2(A, C) \rightarrow \text{Recover} \rightarrow R'(A, B, C)$

Thus, $R' = R$

- Decomposition is lossless if :

$X \cap Y \rightarrow X$, that is: all attributes common to both X and Y functionally determine ALL the attributes in X.

$X \cap Y \rightarrow Y$, that is: all attributes common to both X and Y functionally determine ALL the attributes in Y

If $X \cap Y$ forms a superkey of either X or Y, the decomposition of R is a lossless decomposition.

Dependency Preserving Decomposition

- A decomposition $D = \{R_1, R_2, \dots, R_n\}$ of R is dependency-preserving with respect to F if the union of the projections of F on each R_i in D is equivalent to F;

if $(F_1 \cup F_2 \cup \dots \cup F_n)^+ = F^+$

- Example-

$R = (A, B, C)$

$F = \{A \rightarrow B, B \rightarrow C\}$

Key = {A}

R is not in BCNF

Decomposition $R_1 = (A, B), R_2 = (B, C)$

R_1 and R_2 are in BCNF, Lossless-join decomposition, Dependency preserving

- Each Functional Dependency specified in F either appears directly in one of the relations in the decomposition.
- It is not necessary that all dependencies from the relation R appear in some relation R_i .
- It is sufficient that the union of the dependencies on all the relations R_i be equivalent to the dependencies on R.
- is lost in the decomposition.

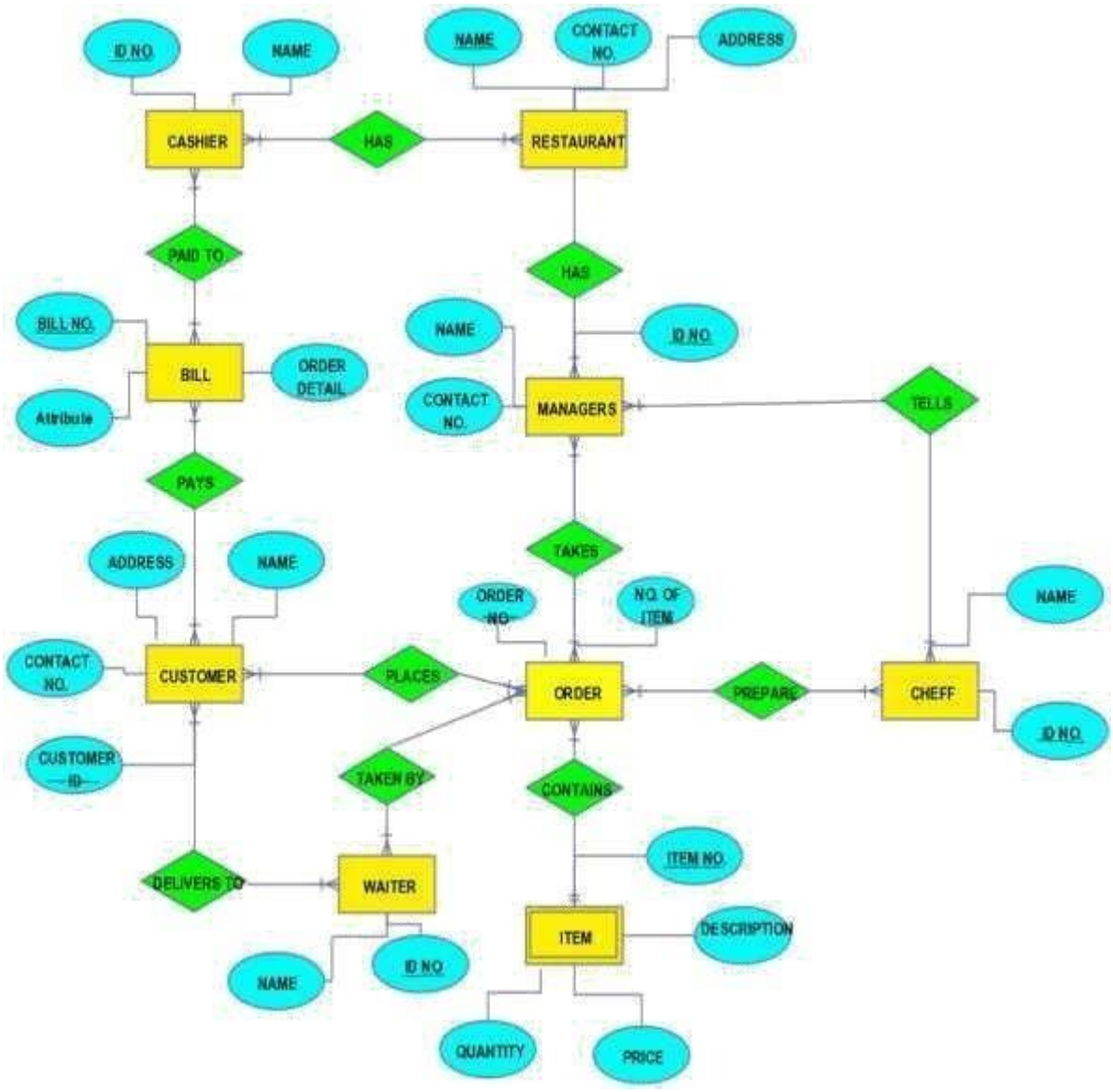
Differences Between E-R Model and Relational Model

1. Difference between E-R Model and Relational Model in DBMS The basic difference between E-R Model and Relational Model is that E-R model specifically deals with entities and their relations. On the other hand, the Relational Model deals with Tables and relation between the data of those tables.
2. An E-R Model describes the data with entity set, relationship set and attributes. However, the Relational model describes the data with the tuples, attributes and domain of the attribute.
3. One can easily understand the relationship among the data in E-R Model as compared to Relational Model.
4. E-R Model has Mapping Cardinality as a constraint whereas Relational Model does not have such constraint.

Example ER Diagrams

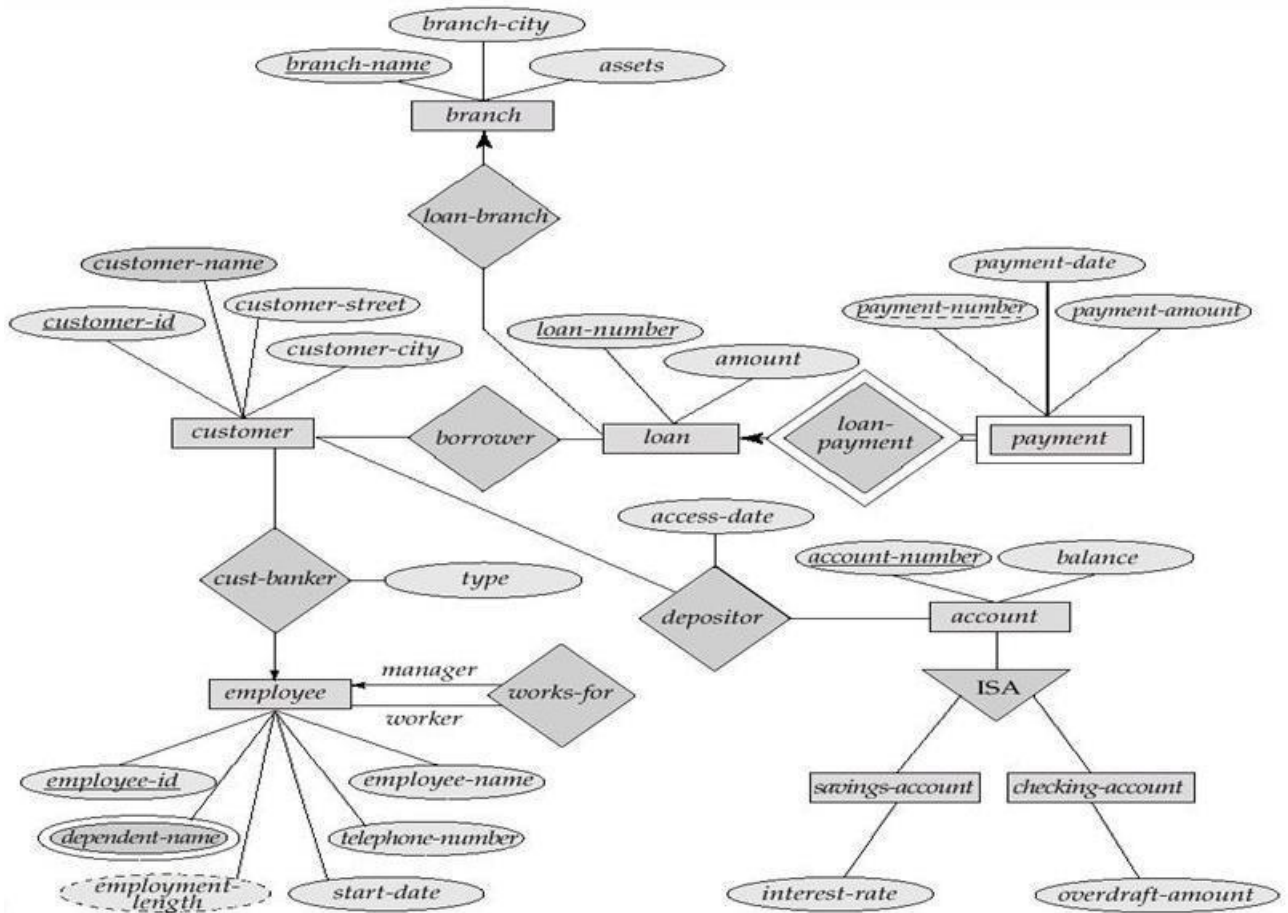
E-R Diagram for Restaurant Menu Ordering System

Draw E-R diagram for restaurant menu ordering System which will facilitate the food items ordering and services within a restaurant. The entire restaurant scenario is detailed as follows. The customer is able to view the food items menu, call the waiter place orders and obtain the final bill through the computer kept in their table. The waiters through their wireless tablet pc are able to initialize a table for customers control the table functions to assist customers, orders send orders to food preparation staff (chef) and finalize the customer bill. The food preparation staff with their touch display interfaces to the system, are able to view orders sent to the kitchen by waiters. During preparation they are able to let the waiter know the status of each item, and can send notification when items are completed. The system should have full accountability and logging facilities, and should support supervisor actions to account for exceptional circumstances, such as a meal being refunded or walked out on? (April/May 2015)

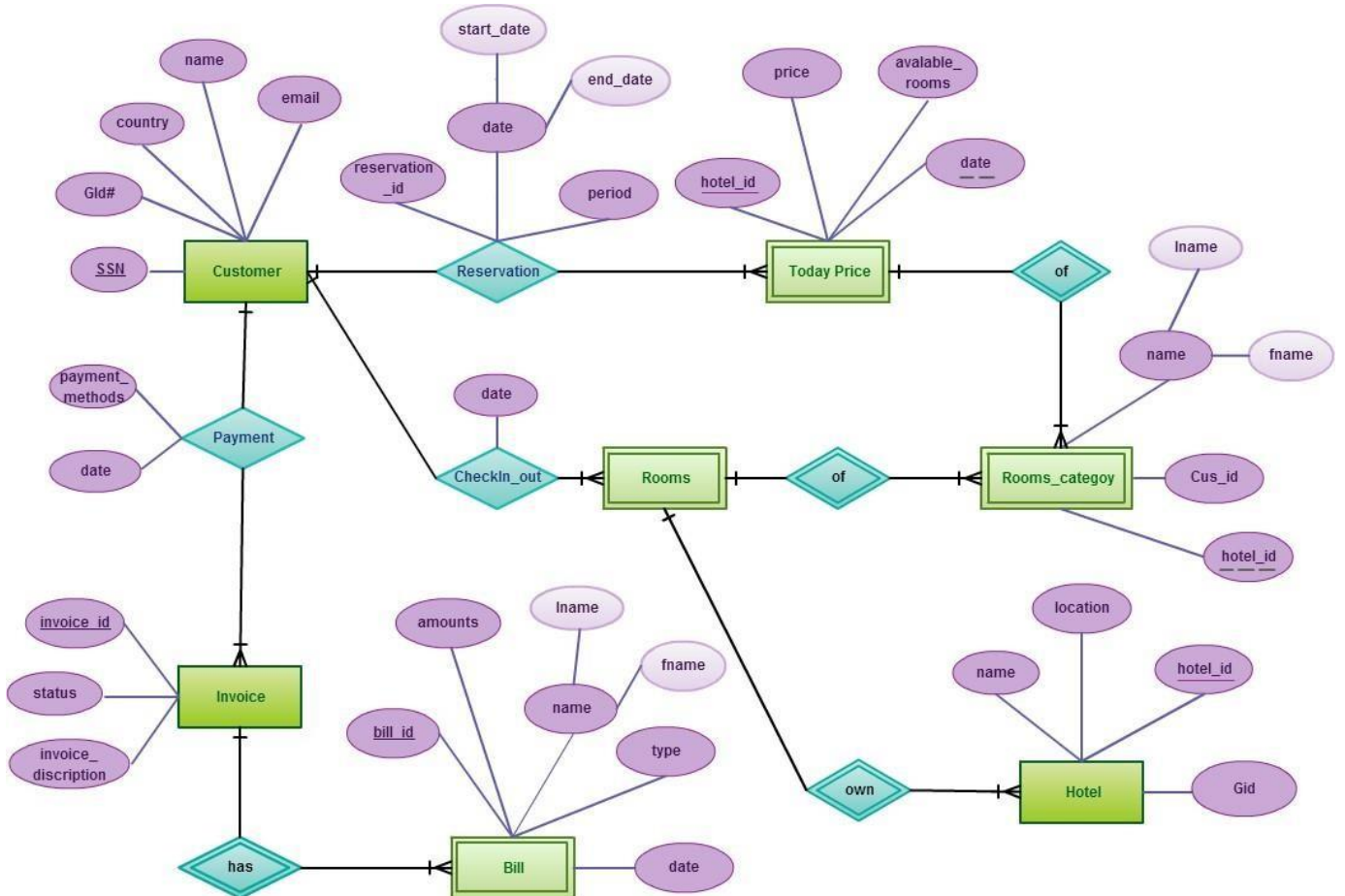


E-R Diagram for Banking Enterprise

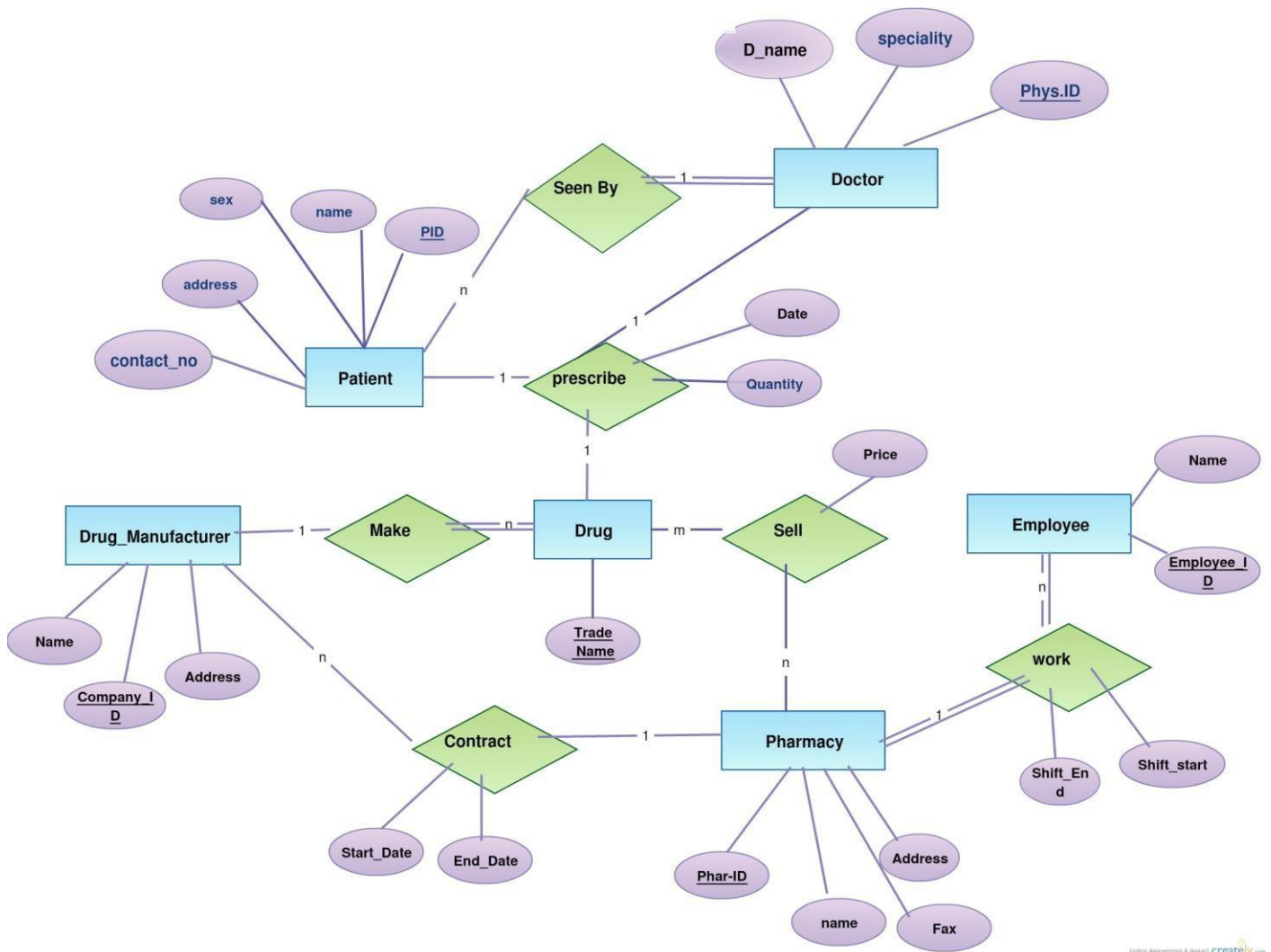
Write short notes on ER diagram for banking enterprise. (Nov/Dec 2014) (Nov/Dec 2017)



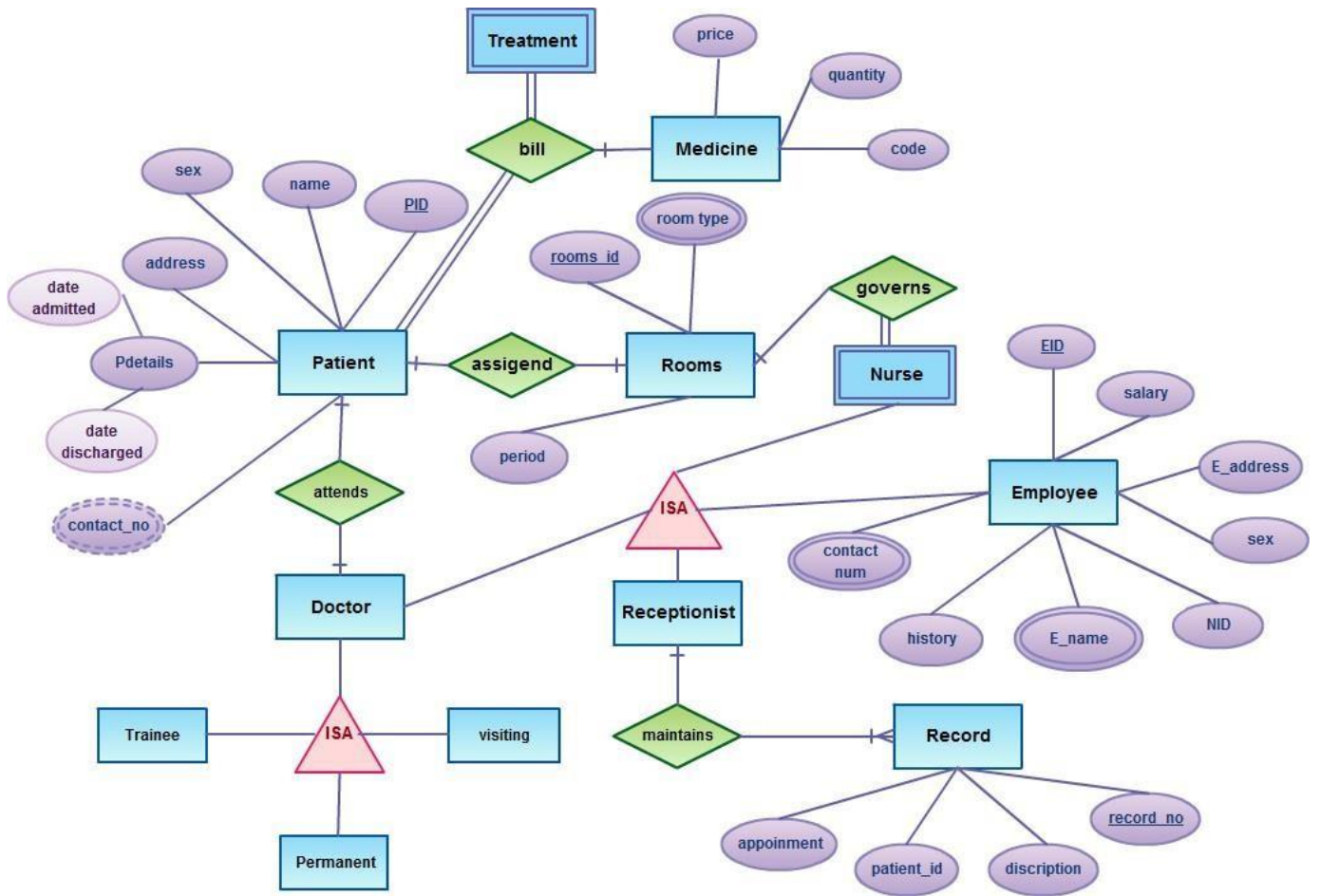
E-R Diagram for Hotel Management System



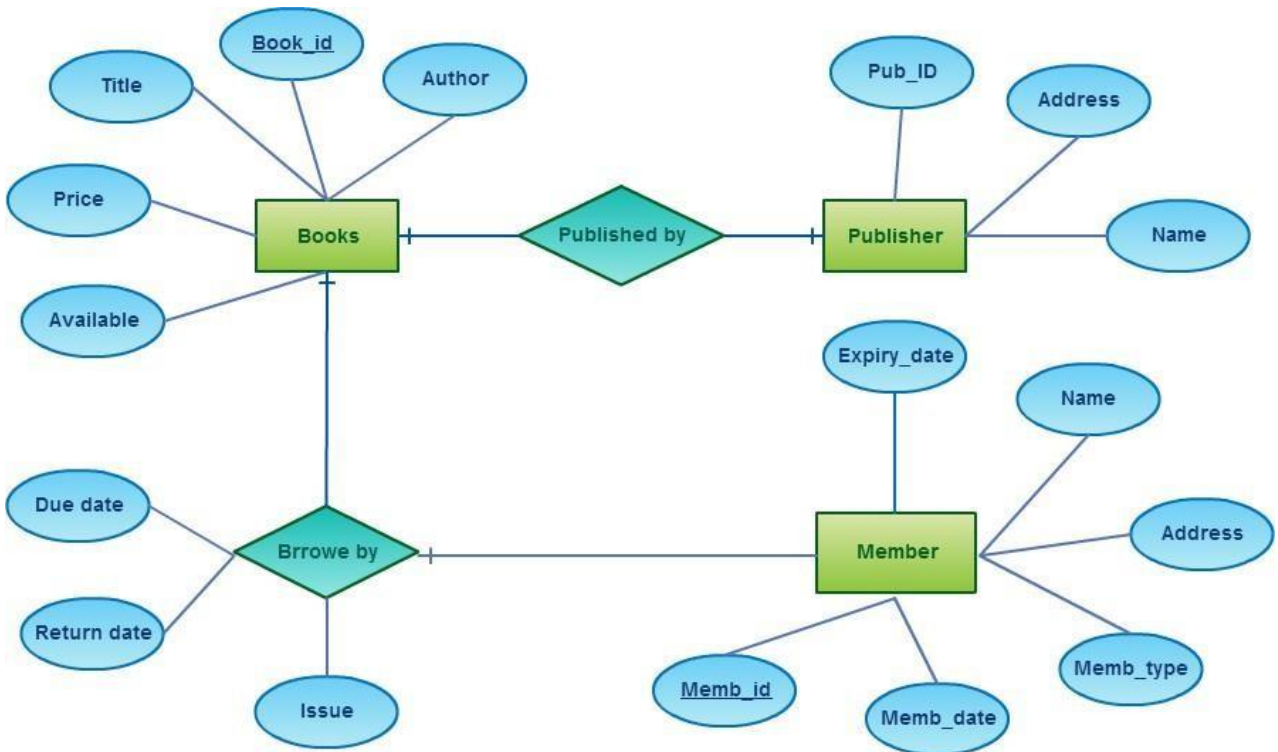
E-R Diagram for Pharmacy Store Information



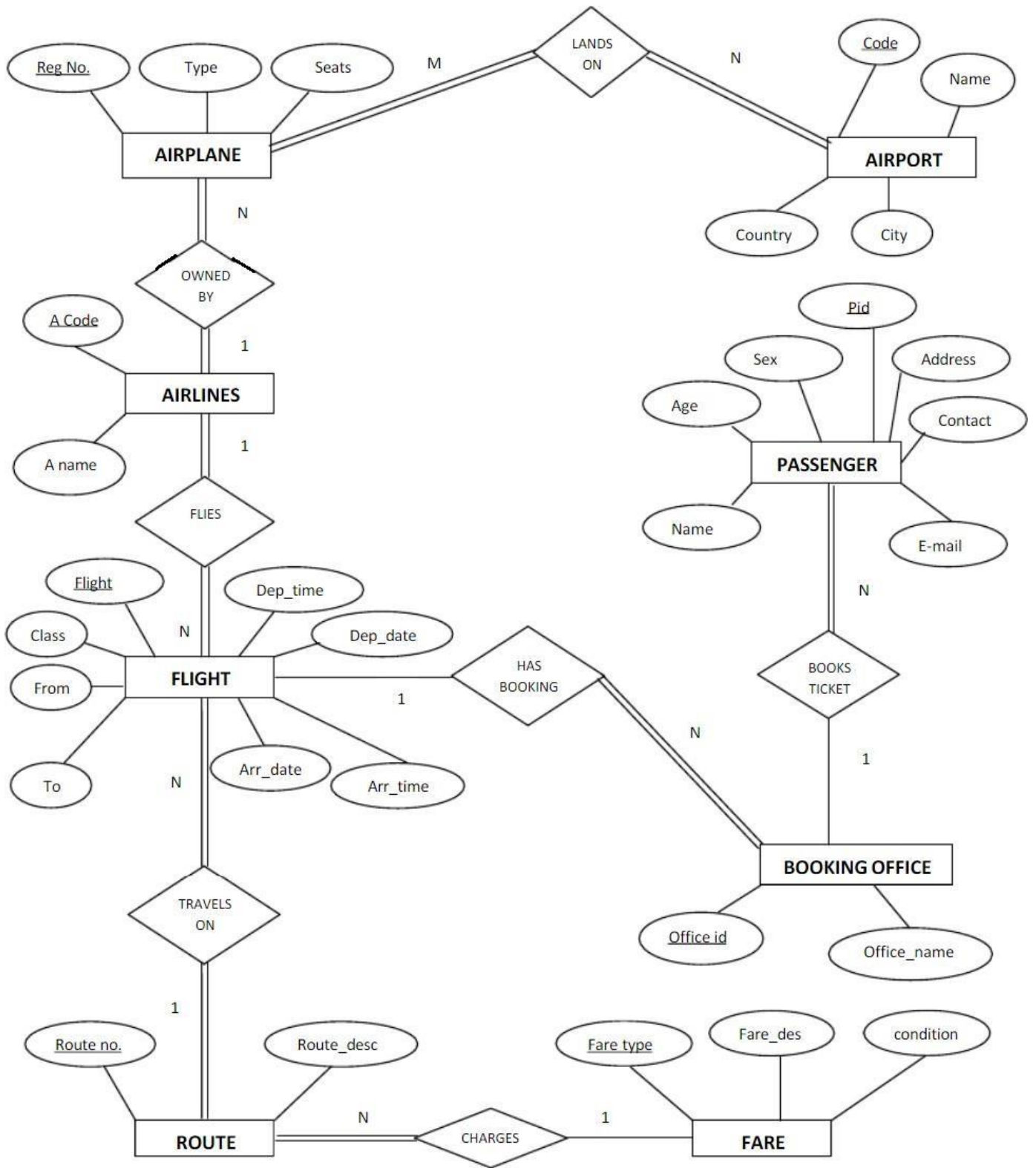
E-R Diagram for Hospital Management System



E-R Diagram for Library Management System

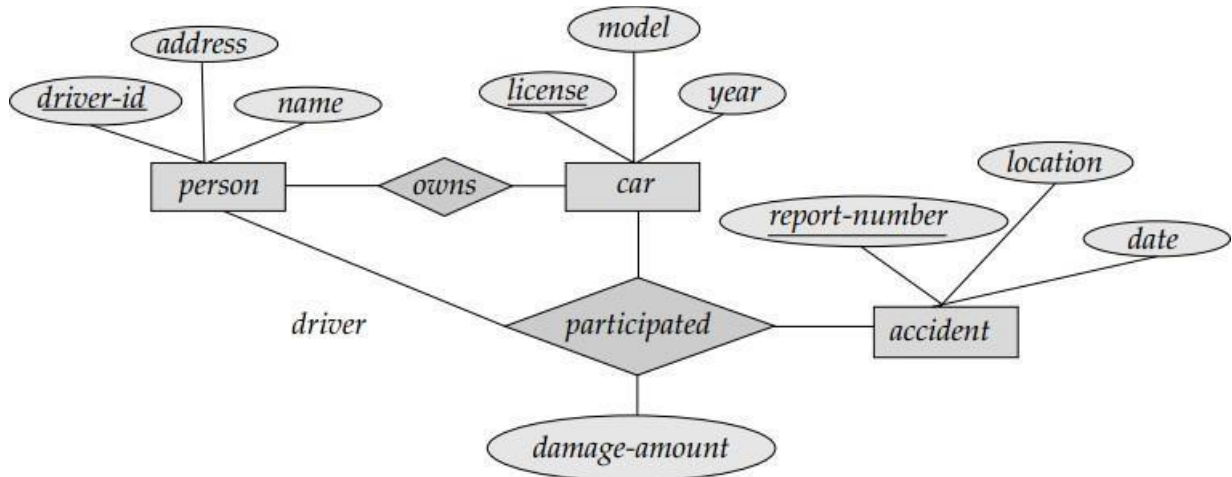


E-R Diagram for Airline Reservation System

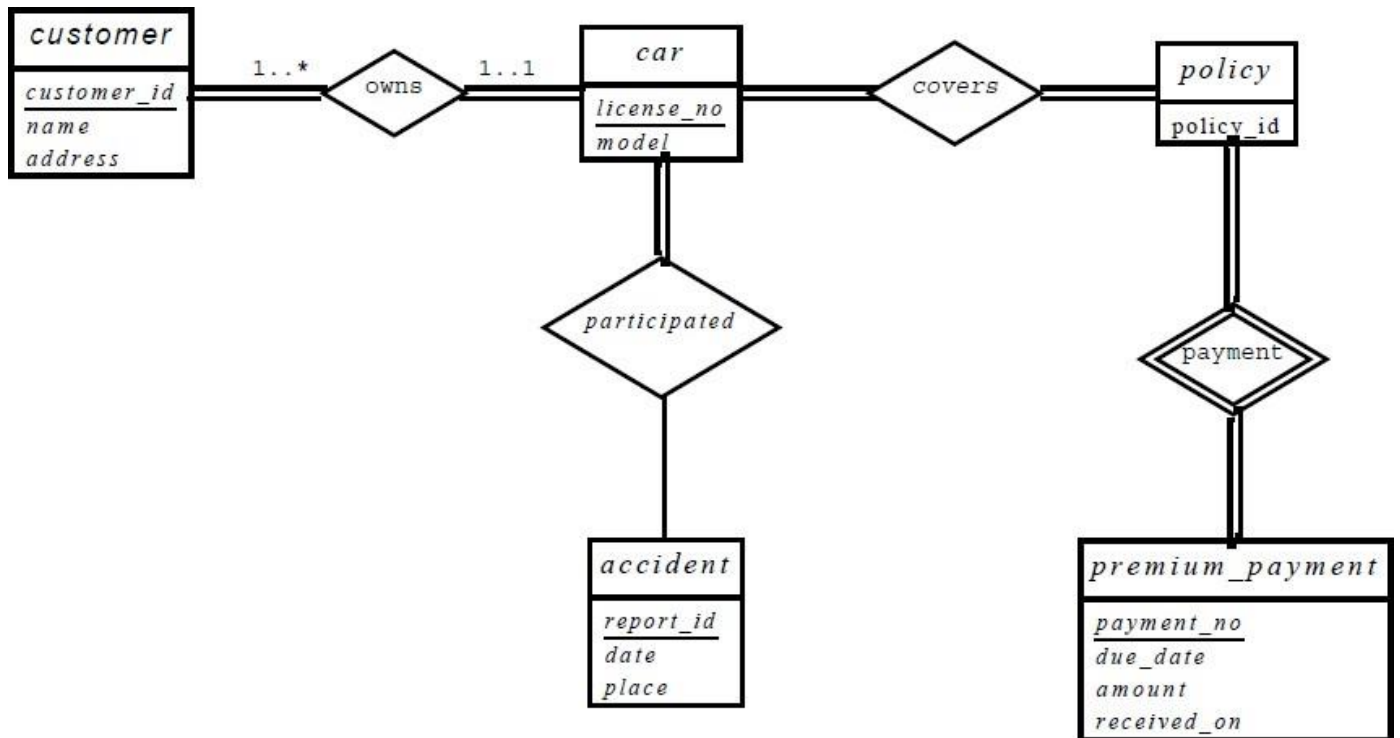


E-R Diagram for Car Insurance Company (Nov/Dec 2016) (Nov/Dec 2018)

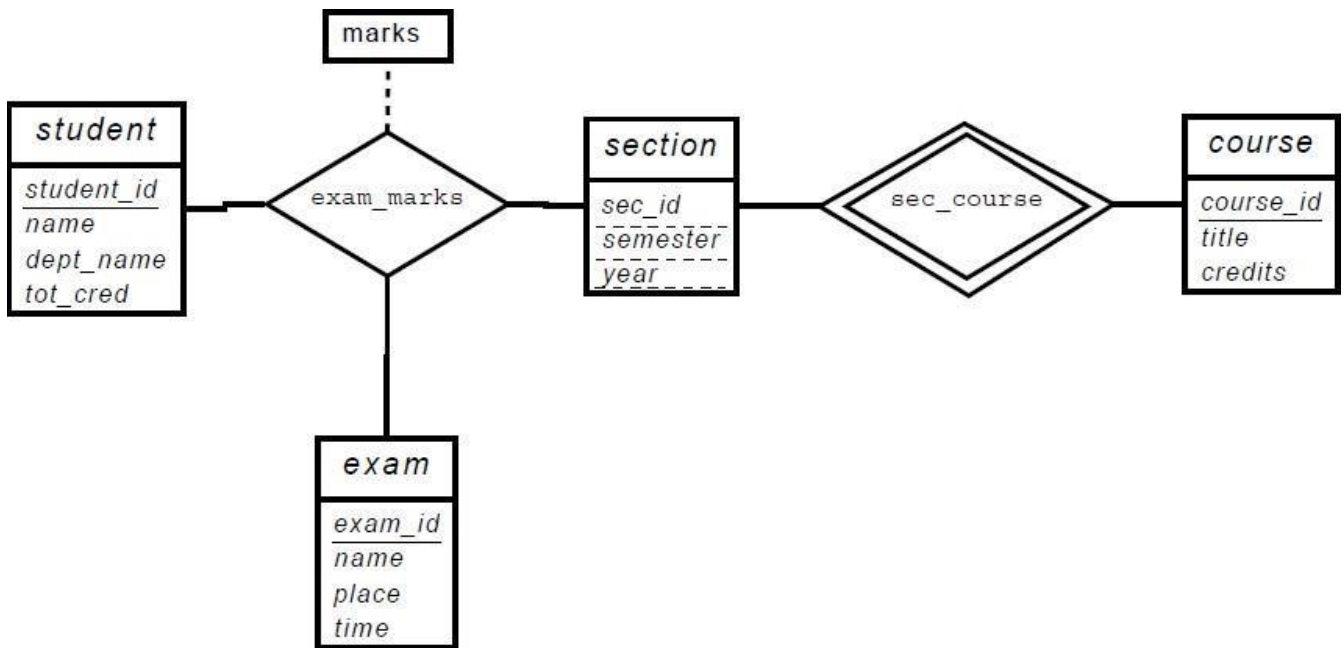
II Year / IV Semester - CSE



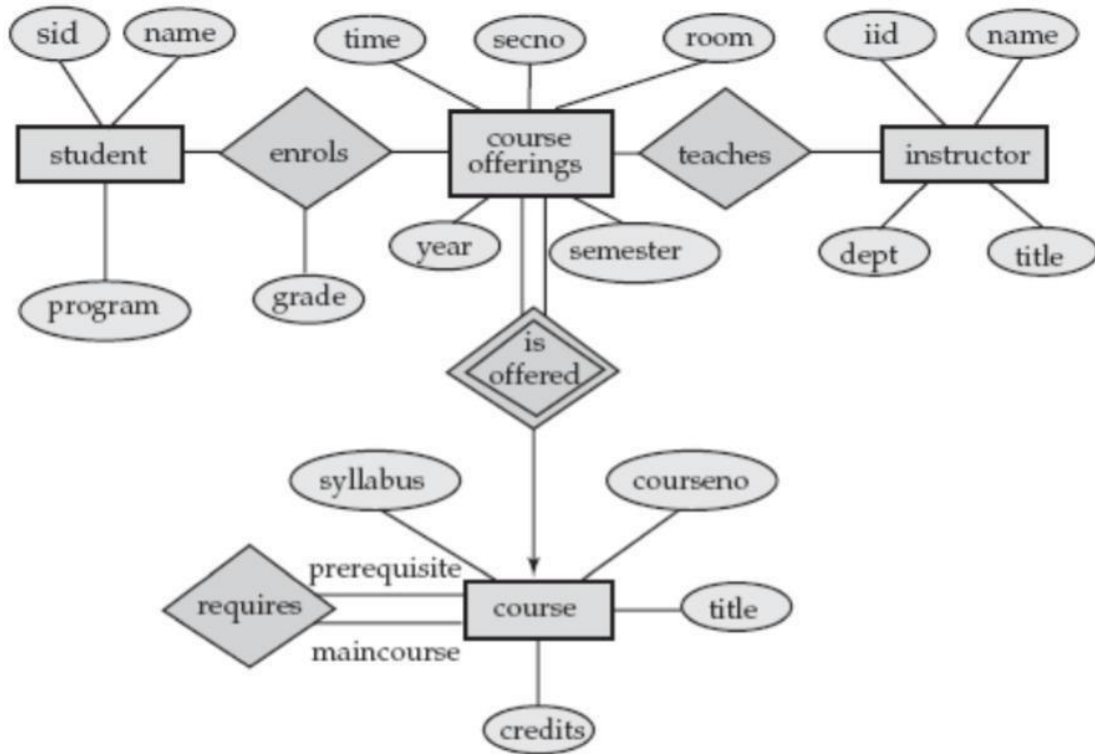
(Or)



E-R Diagram for Marks Database

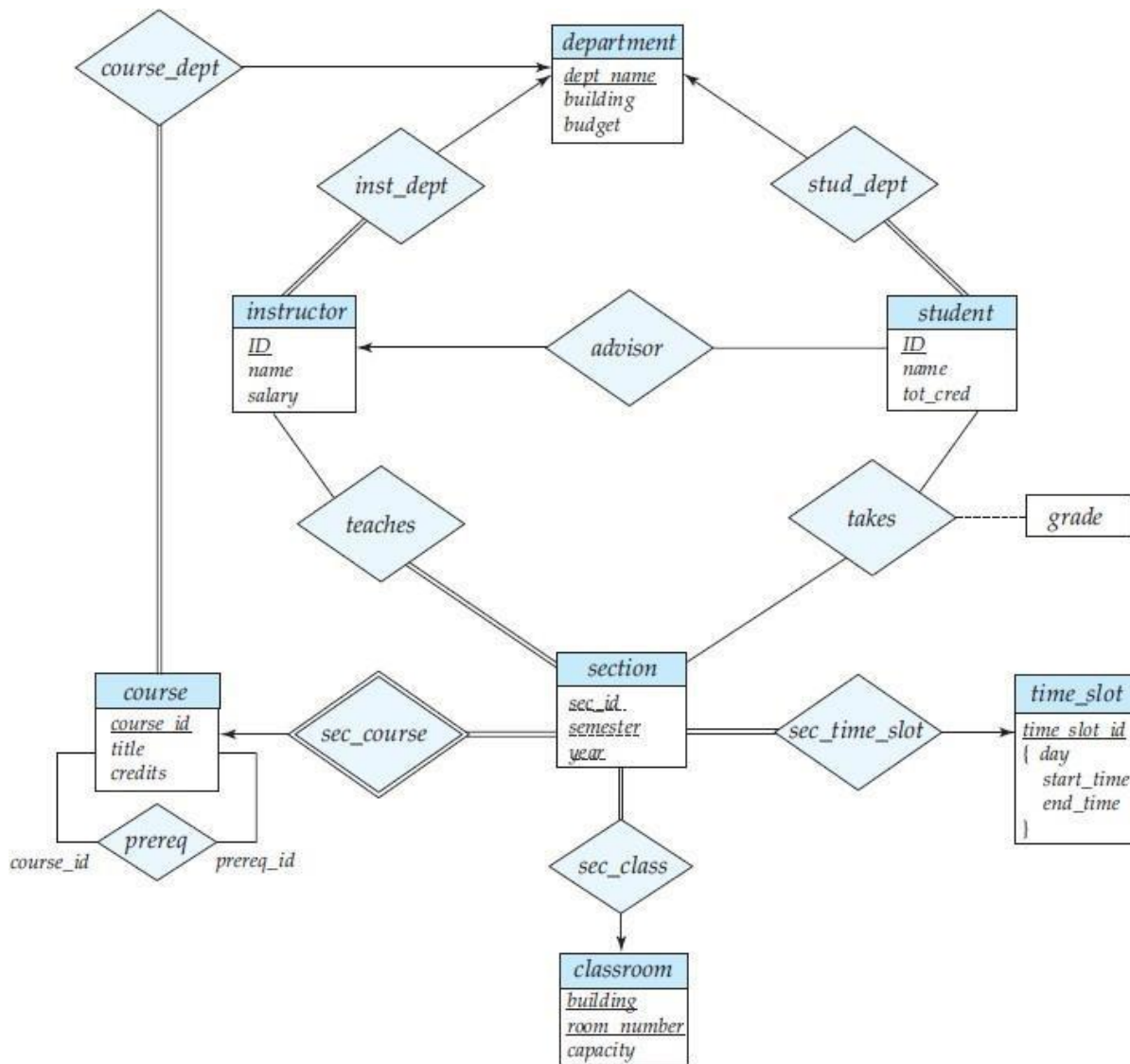


E-R Diagram for University Database (April/May 2018)



(Or)

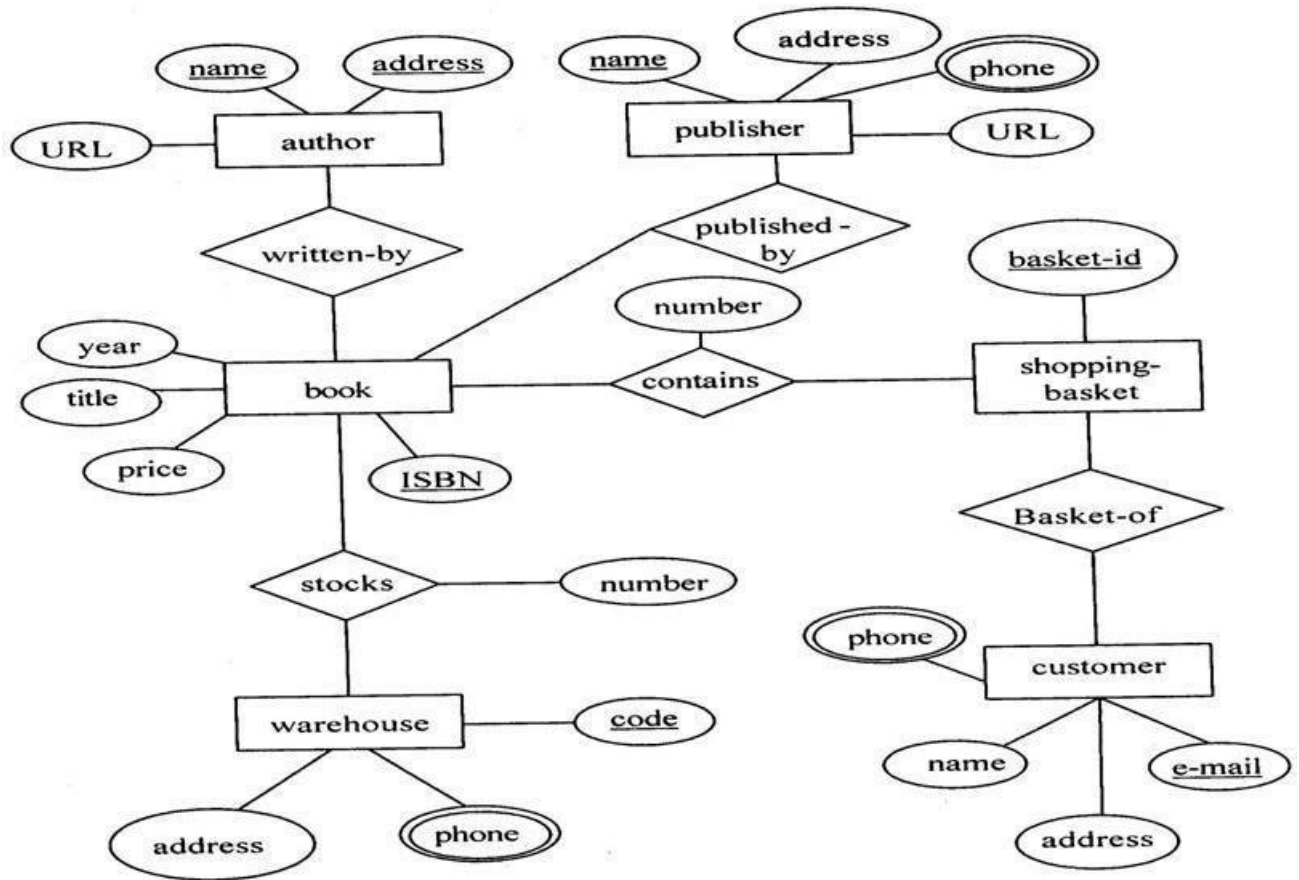
E-R Diagram for the University Enterprise



E-R Diagram for Online Bookstore

Draw an E-R diagram, which models an online bookstore.

- List the entity sets and their primary keys.
- Suppose the bookstore adds Blu-ray discs and downloadable video to its collection. The same item may be present in one or both formats, with differing prices. Extend the E-R diagram to model this addition, ignoring the effect on shopping baskets.
- Now extend the E-R diagram, using generalization, to model the case where a shopping basket may contain any combination of books, Blu-ray discs, or downloadable video.



(Or)

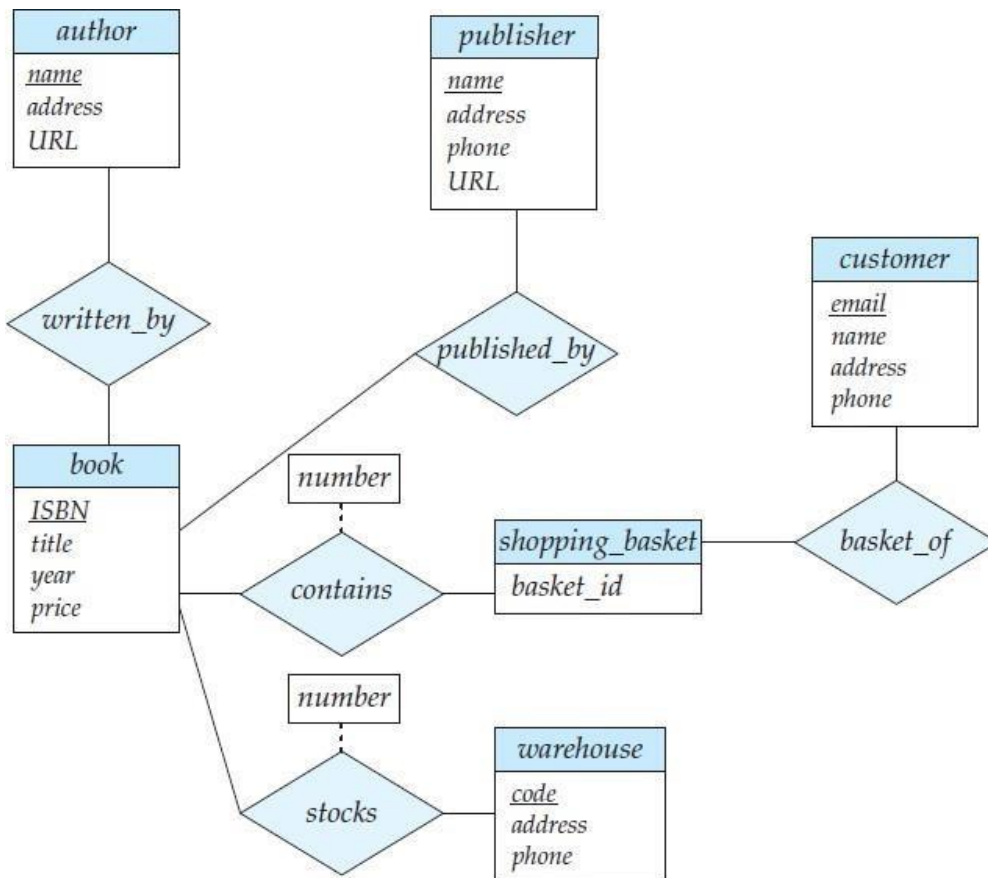
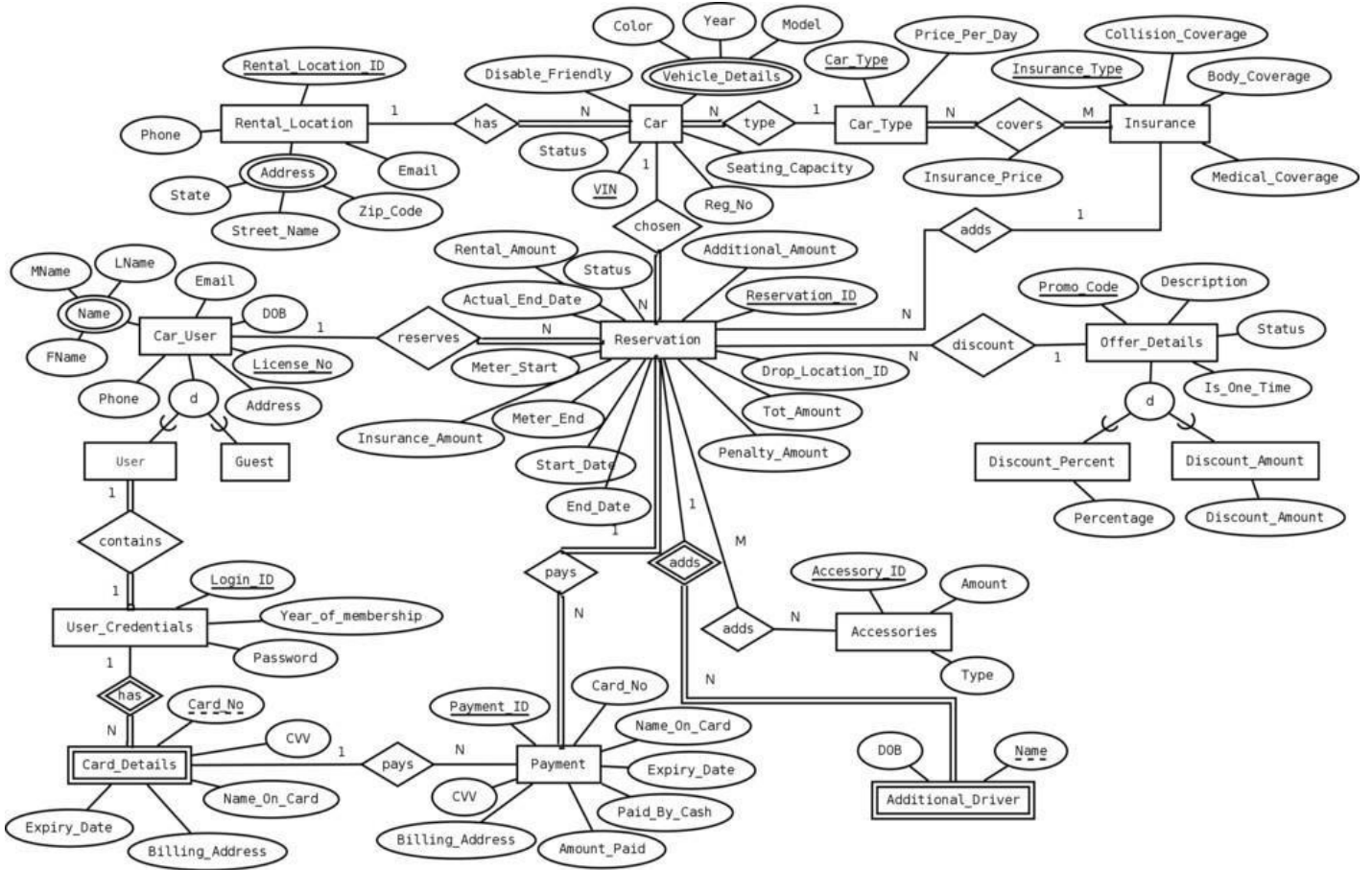


Diagram for Car Rental Company (Nov/Dec 2015)



Unit – I Relational

Databases

Purpose of Database System - Views of Data - Data Models - Database System Architecture - Introduction to Relational Databases - Relational Model - Keys - Relational Algebra - SQL Fundamentals - Advanced SQL Features - Embedded SQL- Dynamic SQL.

Introduction

- Databases and database technology have a major impact on the growing use of computers.
- Databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, genetics, law, education and library science.

Database

What is Database?

- A database is a collection of data elements (facts) stored in a computer in a systematic way.

(Or)

- The collection of data, usually referred to as the **database**, contains information relevant to an enterprise.
- The computer program used to manage and query a database is known as a database management system (DBMS).
- A **Database System (DBS)** is a DBMS together with the data and applications.
- DBMS**: A software package/system that can be used to store, manage and retrieve data from databases.

Database Management System (DBMS)

What is DBMS?

- A **database-management system (DBMS)** is a collection of interrelated data and a set of programs to access those data.
- The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.
- Database systems are designed to manage large bodies of information.
- Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information.

Features of Database:

- It is a persistent (stored) collection of related data.
- The data is input (stored) only once.
- The data is organized (in some fashion).
- The data is accessible and can be queried (effectively and efficiently).

Functionsof DBMS

- ADBMSmakesitpossibleforuserstocreate,editandupdatedataindatabasefiles.
- More specifically, a DBMS provides the following functions:
 - ✓ **Concurrency**: concurrent access (meaning 'at the same time') to the same database by multiple users
 - ✓ **Security**: security rules to determine access rights of users
 - ✓ **Backup and Recovery**: processes to back-up the data regularly and recover data if a problem occurs
 - ✓ **Integrity**: databasestructureandrulesimprove the integrity of thedata
 - ✓ **Data Descriptions**: adata dictionaryprovidesa description of thedata

Database-System Applications**Discuss various database system applications.**

Databases are widely used. Some of the database applications are:

- Enterprise Information**
 - ✓ **Sales**
 - For customer, product, and purchase information.
 - ✓ **Accounting**
 - For payments, receipts, account balances, assets and other accounting information.
 - ✓ **Human Resources**
 - For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.
 - ✓ **Manufacturing**
 - For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.
- Banking and Finance**
 - ✓ **Banking**
 - For customer information, accounts, loans, and banking transactions.
 - ✓ **Credit Card Transactions**
 - For purchases on credit cards and generation of monthly statements.
 - ✓ **Finance**
 - For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.
- Universities**
 - For student information, course registrations, and grades.
- Airlines**
 - For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.

□ **Telecommunication**

- For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

Purpose of Database System

Explain in detail about the purpose of database systems.

- Databasesystems arose in response to early methods of computerizedmanagementof commercial data.
- A **database management system (DBMS)** is a collection of programs that enables users to create and maintain a database.
- The DBMS is a *general-purpose software system* that facilitates the processes of *defining, constructing, manipulating, andsharing*databasesamongvarioususersandapplications.
 - ✓ **Defining** a Database
 - It involves specifying the data types, structures, and constraints of the data to be stored in the database.
 - The database definition or descriptive information is also stored by the DBMS in the form of a databasecatalogordictionary; itiscalled as a**meta-data**.
 - ✓ **Constructing** the Database
 - Itistheprocessofstoringthedataonsomestoragemediumthat iscontrolled by the DBMS.
 - ✓ **Manipulating** a Database
 - Itincludesfunctionssuch as queryingthedatastoretrievespecificdata, updating the database to reflect changes in the miniworld, and generating reports from thedata.
 - ✓ **Sharing** a database
 - Itallowsmultipleusersandprogramstoaccessthedatabasesimultaneously.
- An **application program** accesses the database by sending queries or requests for data to the DBMS.
- A**query** typicallycausessomedatatoberetrieved;a**transaction** maycausessomedatato be read and some data to be written into the database.
- SomeotherimportantfunctionsprovidedbytheDBMSinclude*protecting* thedatabaseand *maintaining* it over a long period of time.
- **Protection** includes,
 - ✓ *System protection* againsthardwareorsoftwaremalfunction(orcashes)
 - ✓ *Security protection* against unauthorized or malicious access
- A typical large database may have a life cycle of many years, so the DBMS must be able to **maintain** the databasesystem byallowingthesystemtoevolvetasrequirementschange over time.

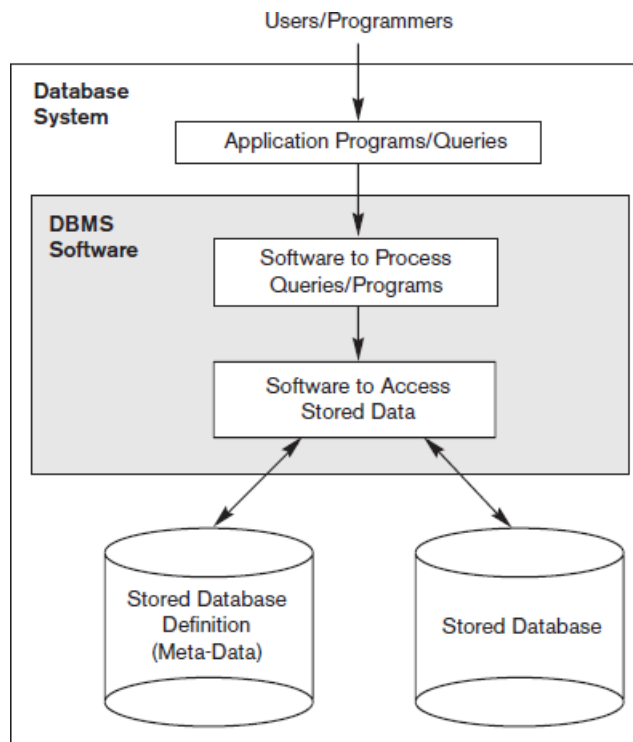


Figure: A Simplified Database System Environment

An Example

- ❑ Let us consider university database, which keeps information about all instructors, students, departments and course offerings.
- ❑ Onewaytokeeptheinformationonacomputeristostoreitinoperatingsystemfiles.
- ❑ To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including programs to:
 - ✓ Add new students, instructors, and courses
 - ✓ Register students for courses and generate class rosters
 - ✓ Assign grades to students, compute grade point averages (GPA), and generate transcripts
- ❑ System programmers wrote these application programs to meet the needs of the university.

File Processing System

Explain in detail about file processing system.

- ❑ It is supported by a conventional operating system.
- ❑ Thesystemstorespermanentrecordsinvariousfiles,anditneedsdifferentapplication programs to extract recordsfrom, andaddrecordsto, theappropriate files.
- Before database management systems (DBMS's) were introduced, organizations usually stored information in such systems.

Disadvantages of File System over DBMS

- ❑ File Processing System has a number of major disadvantages:
 - ✓ Data Redundancy andInconsistency
 - ✓ Difficulty in AccessingData
 - ✓ Data Isolation

- ✓ Integrity Problems
- ✓ Atomicity Problems
- ✓ Concurrent-Access Anomalies
- ✓ Security Problems

Data Redundancy and Inconsistency

- Same information maybe duplicated in several places.
- All copies may not be updated properly.
- Files that represent the same data may become inconsistent.

Difficulty in Accessing Data

- File processing environments do not allow needed data to be retrieved in a convenient and efficient manner.
- Mayhavetowriteanewapplicationprogramtosatisfyanunusualrequest.
- E.g. find all customers with the same postal code.
- Could generate this data manually, but a long job.

Data Isolation

- Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

Integrity Problems

- The data values stored in the database must satisfy certain types of consistency constraints.
- The constraints are enforced by adding appropriate code in programs.
- When new constraints are added it is difficult to change programs to enforce them.

Atomicity Problem

- If any failure occurs the data is to be restored to the consistent state that existed prior to failure.
- It must be atomic happen entirely or not at all.
- It is difficult to ensure atomicity in a conventional file processing system.

Concurrent Access Anomalies

- For the overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously.
- In such an environment, interaction of concurrent updates is possible and may result in inconsistent data.

Security Problems

- Not every user of database system is allowed to access data.
- Ie., Every user of the system should be able to access only the data they are permitted to see.
- Enforcing security constraint is difficult in file processing system.

File Systems vs Database Systems

- DBMS are expensive to create in terms of software, hardware, and time invested.
- The solution is called maintaining data in flat files. So what is bad about flat files?

- ✓ Uncontrolled Redundancy
- ✓ Inconsistent Data
- ✓ Inflexibility
- ✓ Limited Data Sharing
- ✓ Poor Enforcement of Standards
- ✓ Low Programmer Productivity
- ✓ Excessive Program Maintenance
- ✓ Excessive Data Maintenance

File System

- Data is stored in Different Files in forms of Records
- The programs are written time to time as per the requirement to manipulate the data within files.
 - ✓ A program to debit and credit an account
 - ✓ A program to find the balance of an account
 - ✓ A program to generate monthly statements

Advantages of DBMS

- Improved security
- Improved data integrity
- Data consistency
- Improved data accessibility and responsiveness
- Increased concurrency
- Improved backup and recovery services

Disadvantages of DBMS

- Cost of DBMS's
- Complexity and Size
- Higher impact of a failure
- Performance

Characteristics of the Database Approach

Write down the characteristics of database approach.

- The main characteristics of the database approach versus the file-processing approach are the following:
 - ✓ Self-describing nature of a database system
 - ✓ Insulation between programs and data, and data abstraction
 - ✓ Support of multiple views of the data
 - ✓ Sharing of data and multiuser transaction processing

Self-Describing Nature of a Database System

- A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints.

- ❑ This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data.
- ❑ The information stored in the catalog is called **meta-data**, and it describes the structure of the primary database.

Insulation between Programs and Data and Data Abstraction

- ❑ In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require *changing all programs* that access that file.
- ❑ By contrast, DBMS access programs do not require such changes in most cases.
- ❑ The structure of data files is stored in the DBMS catalog separately from the access programs.

Support of Multiple Views of the Data

- ❑ A database typically has many users, each of whom may require a different perspective or **view** of the database.
- ❑ A view may be a subset of the database or it may contain **virtual data** that is derived from the database files but is not explicitly stored.
- ❑ A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.

Sharing of Data and Multiuser Transaction Processing

- ❑ A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time.
- ❑ This is essential if data for multiple applications is to be integrated and maintained in a single database.
- ❑ The DBMS must include **concurrency control** software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.
- ❑ A transaction is an *executing program or process* that includes one or more database accesses, such as reading or updating of database records.

Database Terminologies

List out some of the terminologies used in database.

Some of the terminologies used in databases are, **Database**

- ✓ It is a collection (or list) of information.
- ✓ A database is comprised of one or more lists (called tables) of data organized by columns, rows and cells.

Tables

- ✓ The view displays the database as a combination of rows (records) and columns (fields).

- ✓ The cells contain the bits and pieces of data for each record in each field.
- ✓ The first row of a table is reserved for the field names.

Key

- ✓ A key is a logical value to access record in a table.
- ✓ A key that uniquely identifies a record is called as primary key.

Field Names

- ✓ Identify the different categories in a database.
- ✓ The top row is reserved for field names.
- ✓ Examples of field names are First name, last name, address, city, state, zip, phone number.

Fields

- ✓ It defines the categories in a database.
- ✓ Fields are displayed in columns.
- ✓ For Example, in a database, the zip field contains all the zip codes from each of the records.
- ✓ These are the bits and pieces of data.

Domain

- ✓ Domain refers to the possible values each field can contain.
- ✓ For example (marital status fields may contain either married or unmarried values.)

View

- ✓ It is a virtual table made up of a subset of the actual tables.

Records

- ✓ These are related information that is separated by columns or fields.
- ✓ A name and address are considered one record in the database.
- ✓ A second Name and address are a different record.

Constraints

- ✓ Constraints are the logic rules that are used to ensure data consistency or avoid certain unacceptable operations on the data.

Cells

- ✓ The intersection of columns and rows that contain the data for each record

Index

- ✓ It is the part of the physical structure.

Data

- ✓ All the records of information in a database including the field names.
- ✓ Data + Field Names = Records
- ✓ All Records = a Database

Information

- ✓ Information is data that is processed to have a meaning.

NULL Value

- ✓ A field is said to contain a null value when it contains nothing at all.

Data Integrity

- ✓ It describes the accuracy, validity and consistency of data.

Database Normalization

- ✓ It is a technique that helps to reduce the occurrence of data anomalies and poor data integrity.

Objects

- ✓ Enables you to find, view, display and print data differently, based on your needs.
- ✓ The most commonly used objects are tables, queries, forms and reports.
 - Tables show all records in a spreadsheet format.
 - Queries allow you to ask questions of the one or more tables and show only the information you ask for.
 - Forms display one record at a time.
 - Reports give an organized way of presenting information.

Views of Data**Briefly explain about views of data. (May/June 2016)**

- A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data.
- A major purpose of a database system is to provide users with an *abstract* view of the data.
- That is, the system hides certain details of how the data are stored and maintained.

Data Abstraction

- Data abstraction** generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data.
- For the system to be usable, it must retrieve data efficiently.
- The need for efficiency has led designers to use complex data structures to represent data in the database.
- Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify user's interactions with the system:
 - Physical Level**
 - ✓ The lowest level of abstraction describes *how* the data are actually stored.
 - ✓ The physical level describes complex low-level data structures in detail.
 - Logical Level**
 - ✓ The next-higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data.
 - ✓ The logical level thus describes the entire database in terms of a small number of relatively simple structures.

- ✓ Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as **physical data independence**.
- ✓ Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

type instructor = record

ID : **char** (5);

name : **char** (20);

dept name : **char** (20);

salary : **numeric** (8,2);

end;

- ✓ This code defines a new record type called *instructor* with four fields.
- ✓ Each field has a name and a type associated with it.

□ View Level

- ✓ The highest level of abstraction describes only part of the entire database.
- ✓ Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database.
- ✓ Many users of the database system do not need all this information; instead, they need to access only a part of the database.
- ✓ The view level of abstraction exists to simplify their interaction with the system.
- ✓ The system may provide many views for the same database.

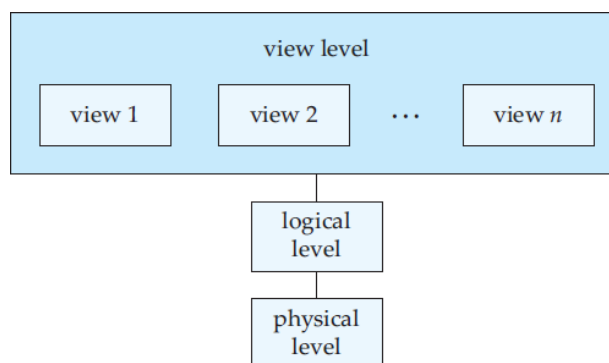


Figure: The Three Levels of Data Abstraction

- A university organization may have several such record types, including
 - ✓ *department*, with fields *dept name*, *building*, and *budget*
 - ✓ *course*, with fields *course id*, *title*, *dept name*, and *credits*
 - ✓ *student*, with fields *ID*, *name*, *dept name*, and *tot cred*
- At the physical level, an *instructor*, *department*, or *student* record can be described as a block of consecutive storage locations. The compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database programmers.

- At the logical level, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction.
- Finally, at the view level, computer users see a set of application programs that hide details of the data types. At the view level, several views of the database are defined, and a database user sees some or all of these views.
- In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database.

Instances and Schemas

Write short notes on instance and schema.

- Databases change over time as information is inserted and deleted.
- The collection of information stored in the database at a particular moment is called an **instance** of the database.
- The overall design of the database is called the database **schema**.
- The concept of database schemas and instances can be understood by analogy to a program written in a programming language.
- A database schema corresponds to the variable declarations (along with associated type definitions) in a program.
- Each variable has a particular value at a given instant.
- The values of the variables in a program at a point in time correspond to an *instance* of a database schema.
- Database systems have several schemas, partitioned according to the levels of abstraction.
- The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level.
- A database may also have several schemas at the view level, sometimes called **subschemas**, that describe different views of the database.

Data Models

Write short notes on data model and its types. (Nov/Dec 2014)

- A collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints.
- A data model provides a way to describe the design of a database at the physical, logical, and view levels.

(Or)

- A **data model** is a collection of concepts that can be used to describe the structure of a database.
- It also includes a set of **basic operations** for specifying retrievals and updates on the database.
- The basic operations provided by the data model, include concepts in the data model to specify the **dynamic aspect** or **behavior** of a database application.

Categories of Data Models

- Data models can be categorized according to the types of concepts they use to describe the database structure.
- High-level or conceptual data models** provide concepts that are close to the way many users perceive Data.
- Low-level or physical data models** provide concepts that describe the details of how data is stored on the computer storage media.
- Representational (or implementation) data models**, which provide concepts that may be easily understood by end users.
- Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.

(Or)

The data models can be classified into four different categories:

Relational Model

- The relational model uses a collection of tables to represent both data and the relationships among those data.
- Each table has multiple columns, and each column has a unique name. Tables are also known as **relations**.
- The relational model is an example of a record-based model.
- Record-based models are so named because the database is structured in fixed-format records of several types.
- Each table contains records of a particular type.
- Each record type defines a fixed number of fields, or attributes.
- The columns of the table correspond to the attributes of the record type.
- The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.

Entity-Relationship Model

- The entity-relationship (E-R) data model uses a collection of basic objects, called *entities*, and *relationships* among these objects.
- An entity is a —thing or —object in the real world that is distinguishable from other objects.
- The entity-relationship model is widely used in database design.

Object-Based Data Model

- Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology.
- This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity.
- The object-relational data model combines features of the object-oriented data model and relational data model.

Semistructured Data Model

- These semistructured data model permits the specification of data where individual data items of the same type may have different sets of attributes.
- This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes.
- The **Extensible Markup Language (XML)** is widely used to represent semistructured data.

Database Languages

Explain in detail about database languages.

- A database system provides a **data-definition language** to specify the database schema and a **data-manipulation language** to express database queries and updates.

Data-Manipulation Language

- A **data-manipulation language (DML)** is a language that enables users to access or manipulate data as organized by the appropriate data model.
- The types of access are:
 - ✓ Retrieval of information stored in the database
 - ✓ Insertion of new information into the database
 - ✓ Deletion of information from the database
 - ✓ Modification of information stored in the database
- There are basically two types:
 - ✓ **Procedural DMLs** require a user to specify *what* data are needed and *how* to get those data.
 - ✓ **Declarative DMLs** (also referred to as **nonprocedural DMLs**) require a user to specify *what* data are needed *without* specifying how to get those data.
- Declarative DMLs are usually easier to learn and use than are procedural DMLs.
- A **query** is a statement requesting the retrieval of information.
- The portion of a DML that involves information retrieval is called a **query language**.
- There are a number of database query languages in use, either commercially or experimentally.
- We study the most widely used query language, SQL.

Data-Definition Language

- We specify a database schema by a set of definitions expressed by a special language called a **data-definition language (DDL)**.
- The DDL is also used to specify additional properties of the data.
- We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a **data storage and definition** language.
- These statements define the implementation details of the database schemas, which are usually hidden from the users.
- The data values stored in the database must satisfy certain **consistency constraints**.

- For example, suppose the university requires that the account balance of a department must never be negative.
- The DDL provides facilities to specify such constraints.
- In general, a constraint can be an arbitrary predicate pertaining to the database.

Domain Constraints

- A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types).
- Declaring an attribute to be of a particular domain acts as a constraint on the values that it can take.
- Domain constraints are the most elementary form of integrity constraint.
- They are tested easily by the system whenever new data items are entered into the database.

Referential Integrity

- There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity).
- For example, the department listed for each course must be one that actually exists.
- More precisely, the *dept name* value in a *course* record must appear in the *dept name* attribute of some record of the *department* relation.
- Database modifications can cause violations of referential integrity.
- When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation.

Assertions

- An assertion is any condition that the database must always satisfy.
- Domain constraints and referential-integrity constraints are special forms of assertions.
- However, there are many constraints that we cannot express by using only these special forms.
- For example, —Every department must have at least five courses offered every semester— must be expressed as an assertion.
- When an assertion is created, the system tests it for validity.
- If the assertion is valid, then any future modification to the database is allowed only if it does not cause that assertion to be violated.

Authorization

- We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database.
- These differentiations are expressed in terms of **authorization**.
 - ✓ **Read Authorization** - It allows reading, but not modification of data.
 - ✓ **Insert Authorization** - It allows insertion of new data, but not modification of existing data
 - ✓ **Update Authorization** - It allows modification, but not deletion of data.

- ✓ **Delete Authorization** - It allows deletion of data.

Database Designers

- ❑ **Database designers** are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.
- ❑ These tasks are mostly undertaken before the database is actually implemented and populated with data.
- ❑ It is the responsibility of database designers to communicate with all prospective database users in order to understand their requirements and to create a design that meets these requirements.

Database Design for a University Organization

- ❑ Let us examine how a database for a university could be designed.
- ❑ The university is organized into departments. Each department is identified by a unique name (*dept name*), is located in a particular *building*, and has a *budget*.
- ❑ Each department has a list of courses it offers. Each course has associated with it a *course id*, *title*, *dept name*, and *credits*, and may also have associated *prerequisites*.
- ❑ Instructors are identified by their unique *ID*. Each instructor has *name*, associated department (*dept name*), and *salary*.
- ❑ Students are identified by their unique *ID*. Each student has a *name*, an associated major department (*dept name*), and *tot cred* (total credit hours the student earned thus far).
- ❑ The university maintains a list of classrooms, specifying the name of the *building*, *room number*, and *room capacity*.
- ❑ The university maintains a list of all classes (sections) taught. Each section is identified by a *course id*, *sec id*, *year*, and *semester*, and has associated with it a *semester*, *year*, *building*, *room number*, and *time slot id* (the time slot when the class meets).
- ❑ The department has a list of teaching assignments specifying, for each instructor, the sections the instructor is teaching.
- ❑ The university has a list of all student course registrations, specifying, for each student, the courses and the associated section that the student has taken (registered for).

Record Based Data Models

Relational Data Model

- ❑ The relational model uses a collection of tables to represent both data and the relationships among those data.
- ❑ Each table has multiple columns, and each column has a unique name.
- ❑ The data is arranged in a relation which is visually represented in a two dimensional table.
- ❑ The data is inserted into the table in the form of tuples (which are nothing but rows).
- ❑ A tuple is formed by one or more than one attributes, which are used as basic building blocks in the formation of various expressions that are used to derive meaningful information.

- The relational model is implemented in databasewhere,
 - ✓ A relation is represented by atable.
 - ✓ A tuple is represented by a row.
 - ✓ An attribute is represented by a column of the table.
 - ✓ Attributenameisthenameofthecolumnsuchas_Identifier‘, _name‘, _city‘etc.,
 - ✓ Attribute valuecontains the valuefor columnin the row.
- It is example for record based model because the database is structured in fixed format records of several types.

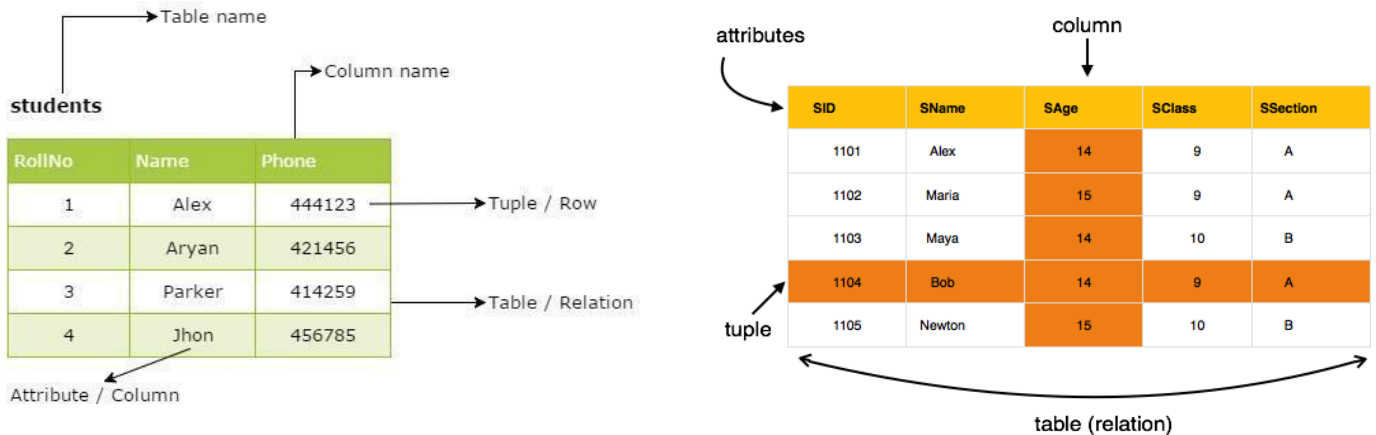


Figure: Relational Model

Network Data Model

- In network model the data are represented by collection of records and their relationship is represented bylinks.
- Network database consists of collection of records connected to one another through links.
- Eachrecord is a collectionoffields or attributes& each of whichcontainonlyonedata value.
- A link is an associated between two records.

Example:

Customer record is defined as,

Type customer = **record**

Customer_name:string;
 Customer_street:string;
 Customer_city:string;

End

- Account records is defined as,

Type account = **record**

Acc_number : string;
 Balance :integer;

End

- In network model the two records are represented as,

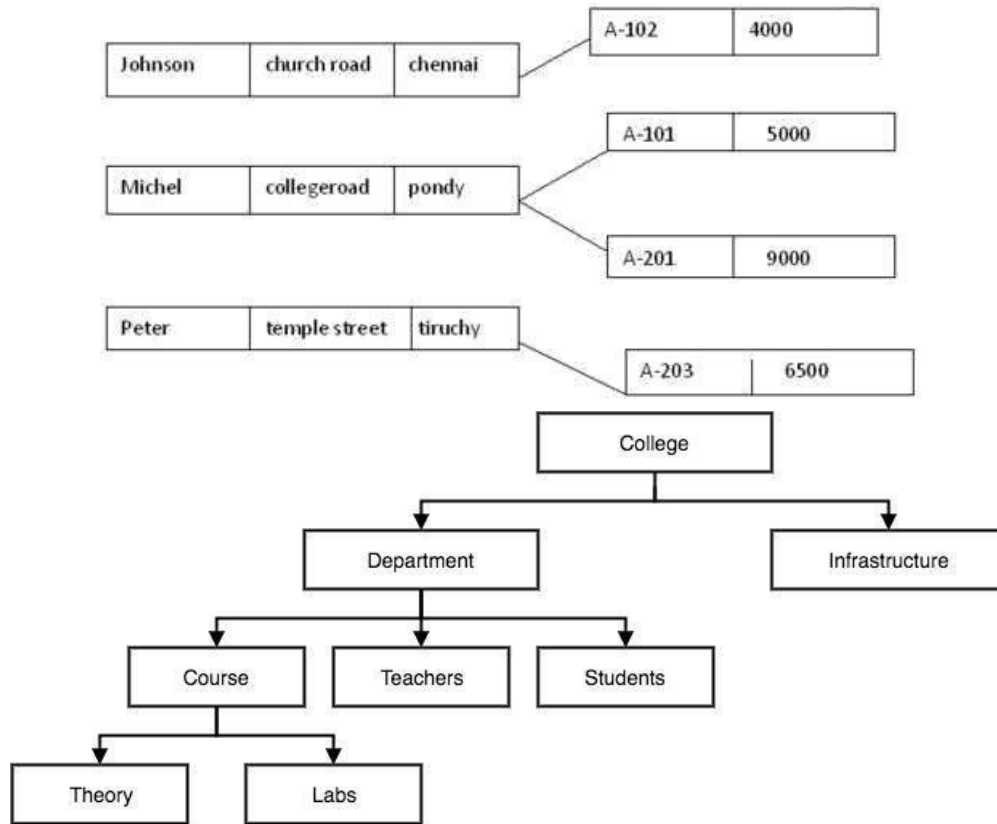


Figure: Network Model

Hierarchical Data Model

- ❑ Hierarchical model consists of a collection of records that are connected to each other through links records are organized as a collection of trees.
- ❑ The hierarchical database model looks like an organizational chart or a family tree. It has a single root segment (Employee) connected to lower level segments (Compensation, Job Assignments and Benefits).
- ❑ Each subordinate segment in turn, may connect to other subordinate segments.
- ❑ Here, compensation connects to Performance Ratings and Salary History.
- ❑ Benefits connect to Pension, Life Insurance and Health.
- ❑ Each subordinate segment is the child of the segment directly above it.

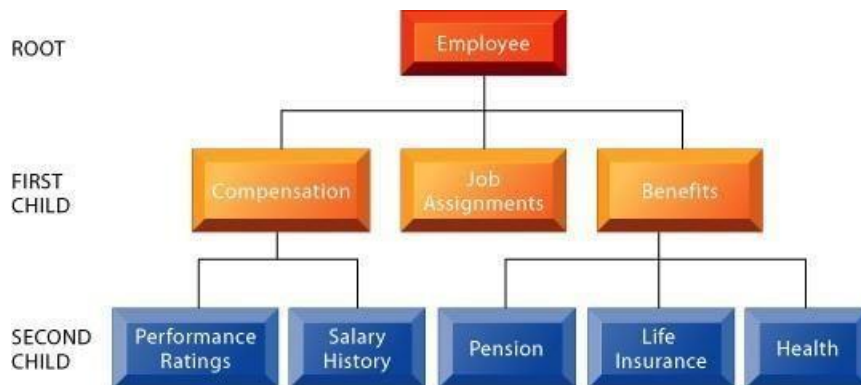


Figure: A Hierarchical data model for Human Resource System

Object-Oriented Data Model / Object Based Data Model

- ❑ The data is stored in the form of objects, which are structures called *classes* that display the data within.

- The fields are instances of these classes.
- Object oriented data model is extending the E-R model with notions of encapsulation, methods and object identity.
- Object oriented data model also supports a rich type system including structured and collection types.
- Object relational data model, a data model that combines features of the object-oriented data model and relational data model.

Semi-structured Data Model

- Semi structured data models** permit the specification of data where individual data items of the same type may have different sets of attributes.
- This is in contrast with the data models mentioned earlier, where every data item of a particular type must have the same set of attributes.
- The extensible markup language (XML)** is widely used to represent semi-structured data.

Database System Architecture

With help of a neat block diagram explain the basic architecture of a database management system. (Nov/Dec 2015) (Or) Briefly explain about database system architecture. (May/June 2016) (Or) State and explain the architecture of DBMS. (Nov/Dec 2017)

- The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs.
- Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines.

Components of DBMS

Explain the components of database in detail.

- Database system is partitioned into modules that deal with each of the responsibilities of the overall system.
- The functional components of the database system are,
 - ✓ Storage Manager
 - ✓ Query Processor

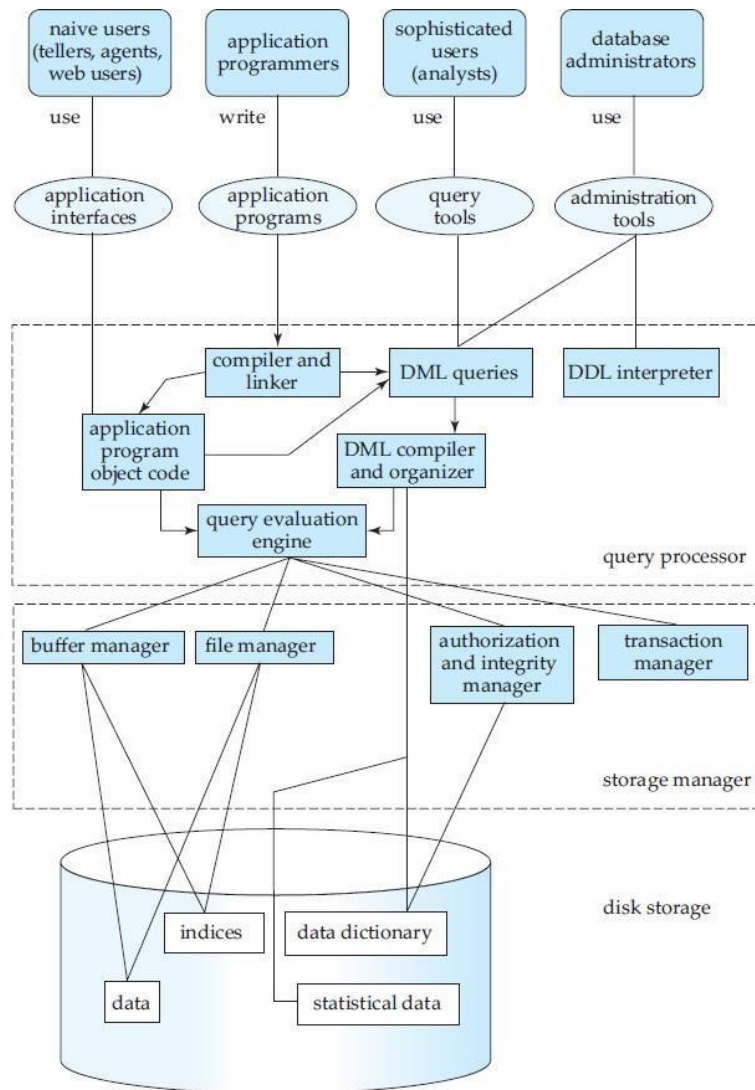


Figure: Database System Structure

Storage Manager

- It is a component of database system that provides the interface between the low- level data stored in the database and the application programs and queries submitted to the system.
- It is responsible for the interaction with the file manager.
- The raw data are stored on the disk using the file system provided by the operating system.
- The storage manager translates the various DML statements into low-level file-system commands.
- The components of storage manager are,
 - ✓ Authentication & Integrity Manager
 - ✓ File Manager
 - ✓ Buffer Manager
 - ✓ Transaction Manager

Authorization & Integrity Manager

- It tests for satisfaction of integrity constraints and checks the authority of users to access data.

File Manager

- It manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

Buffer Manager

- It is responsibility for fetching data from disk storage to main memory & deciding what data to cache in main memory.

Transaction Manager

- It ensures that database remains in a consistent state despite system failure and the concurrent transaction executions proceed without conflicting.
- It consists of the concurrency control manager and the recovery manager.
- The Four Properties of Transactions are,
 - ✓ **Atomicity**
 - This means that either all of the instructions within the transaction will be reflected in the database, or none of them will be reflected.
 - ✓ **Consistency**
 - If we execute a particular transaction in isolation or together with other transaction, (i.e. imagine in a multi-programming environment), the transaction will yield the same expected result.
 - ✓ **Isolation**
 - In case multiple transactions are executing concurrently and trying to access a sharable resource at the same time, the system should create an ordering in their execution so that they should not create any anomaly in the value stored at the sharable resource.
 - ✓ **Durability**
 - It states that once a transaction has been completed, the changes it has made should be permanent.
- The storage manager implements several data structure such as part of the physical system implementation:
 - ✓ Data Files – Stores the database itself.
 - ✓ Data Dictionary- Stores the metadata about the structure of the database (i.e) Schema of the database.
 - ✓ Indices- It provides fast access to data items. The Database provides pointers to those data items that hold a particular index value.

Query Processor

- It helps the database system to simplify and facilitate access to data components of query processor.
- The Components of query processor includes:
 - ✓ DDL Interpreter
 - ✓ DML Compiler

- ✓ Query Optimization
- ✓ Query Evaluation Engine

DDL Interpreter

- Interprets DDL statements and records the definitions in a data dictionary.

DML Compiler

- Translates DML statements into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
- It also performs query optimization (i.e) it picks the lowest cost evaluation plan from among the alternatives.

Query Evaluation Engine

- Executes low-level instructions generated by the DML compiler.
- Database systems can be centralized as client-server.
- Based on this database applications are portioned into
 - ✓ Two Tier Architecture
 - ✓ Three tier Architecture

Two-tier Architecture

- Application resides at the client machine where it invokes database system functionality at the server machine through query language statements.
- E.g. client programs using ODBC/JDBC to communicate with a database.

Three-tier Architecture

- Client machine acts as merely a front end and does not contain any direct database calls.
- The client end communicates with an application server through forms interface and the application server in turn communicates with the database system to access data.
- E.g. web-based applications and applications built using —middleware.

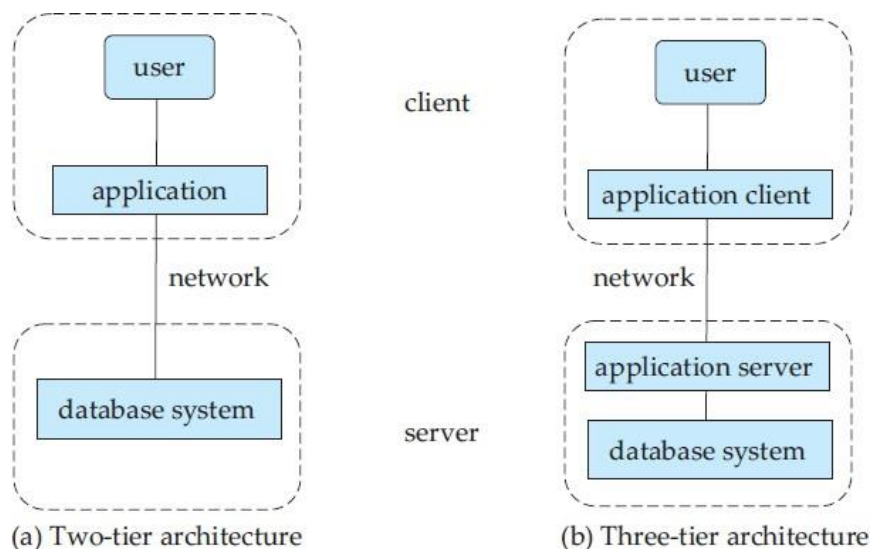


Figure: Two-tier and Three-tier Architectures

Transaction Management**Write short notes on transaction management.**

- A **transaction** is a collection of operations that performs a single logical function in a database application.
- Each transaction is a unit of both atomicity and consistency.
- It is the programmer's responsibility to define properly the various transactions, so that each preserves the consistency of the database.
- For example, the transaction to transfer funds from the account of department *A* to the account of department *B* could be defined to be composed of two separate programs: one that debits account *A*, and another that credits account *B*.
- The execution of these two programs one after the other will indeed preserve consistency.
- Clearly, it is essential that either both the credit and debit occur, or that neither occur.
- That is, the funds transfer must happen in its entirety or not at all. This all-or-none requirement is called **atomicity**.
- In addition, it is essential that the execution of the funds transfer preserve the consistency of the database.
- That is, the value of the sum of the balances of *A* and *B* must be preserved.
- This correctness requirement is called **consistency**.
- Finally, after the successful execution of a funds transfer, the new values of the balances of accounts *A* and *B* must persist, despite the possibility of system failure. This persistence requirement is called **durability**.

Recovery Manager

- It is the responsibility of the database system itself especially, the recovery manager to ensuring the atomicity and durability properties.

Failure Recovery

- It detects system failures and restores the database to the state that existed prior to the occurrence of the failure.

Concurrency-Control Manager

- It is responsible to control the interaction among the concurrent transactions, to ensure the consistency of the database.
- The **transaction manager** consists of the concurrency-control manager and the recovery manager.

Database Users and Administrators

- A primary goal of a database system is to retrieve information from and store new information into the database.
- There are four different types of database-system users, differentiated by the way they expect to interact with the system.
- Different types of user interfaces have been designed for the different types of users.

Naive Users

- They are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously.

Application Programmers

- They are computer professionals who write application programs.
- Application programmers can choose from many tools to develop user interfaces.
- Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports with minimal programming effort.

Sophisticated Users

- They interact with the system without writing programs.
- Instead, they form their requests either using a database query language or by using tools such as data analysis software.

Specialized Users

- They are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework.
- Among these applications are computer-aided design systems, knowledgebase and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

Database Administrators

- In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software.
- Administering these resources is the responsibility of the **database administrator (DBA)**.
- The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed.

(Or)

- DBMS's do not have central control of both the data and the programs that access those data.
- A person who has such central control over the system is called a **database administrator (DBA)**.
- The functions of a DBA include:
 - Schema Definition**
 - ✓ The DBA creates the original database schema by executing a set of data definition statements in the DDL.
 - Storage Structure and Access-method Definition**
 - Schema and Physical-organization Modification**
 - ✓ The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.
 - Granting of Authorization for Data Access**

- ✓ By granting different types of authorization, the database administrator can regulate which parts of the database various users can access.
- ✓ The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.

Routine Maintenance

- Examples of the database administrator's routine maintenance activities are:
 - ✓ Periodically backing up the database, either on tapes or on remote servers, to prevent loss of data in case of disasters such as flooding.
 - ✓ Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
 - ✓ Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

Introduction to Relational Databases

Explain relational DBMS in detail.

- A Relational Database management System (RDBMS) is a database management system based on relational model introduced by E.F Codd.
- In relational model, data is represented in terms of tuples (rows).
- RDBMS is used to manage Relational database.
- Relational database is a collection of organized set of tables from which data can be accessed easily.
- Relational Database is most commonly used database.
- It consists of number of tables and each table has its own primary key.
- RDBMSs are a common choice for the storage of information in new databases used for financial records, manufacturing and logistical information, personnel data and other applications since the 1980s.
- A data model is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints.
- A relational database is based on the relational model which uses a collection of tables to represent both data and the relationships among those data.
- It also includes a DML and DDL.
- A software system used to maintain relational databases is a relational database management system (RDBMS).
- Virtually all relational database systems use SQL (Structured Query Language) for querying and maintaining the database.

Relational Model

- The relational model is today the primary data model for commercial data processing applications.
- This model organizes data into one or more tables (or "relations") of columns and rows, with a unique key identifying each row.

- Rows are also called records or tuples. Columns are also called attributes.
- Generally, each table/relation represents one "entity type" (such as customer or product).
- The rows represent instances of that type of entity (such as "Lee" or "chair") and the columns representing values attributed to that instance (such as address or price).

Keys

Write short notes on keys in DBMS.

- It is defined as one or more columns in a database table that is used to sort and/or identify rows in a table.
- e.g. if you were sorting people by the field salary then the salary field is the key.
- It also establishes relationship among tables.

Types of keys in DBMS

- **Primary Key** – A primary is a column or set of columns in a table that uniquely identifies tuples (rows) in that table. The primary key cannot be null (blank). The primary key is indexed.
- **Super Key** – A super key is a set of one or more columns (attributes) to uniquely identify rows in a table.
- **Candidate Key** – A super key with no redundant attribute is known as a candidate key.
- **Alternate Key** – Out of all candidate keys, only one gets selected as primary key, remaining keys are known as alternate or secondary keys.
- **Composite Key** – A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called a composite key.
- **Foreign Key** – Foreign keys are the columns of a table that point to the primary key of another table. They act as a cross-reference between tables.

Relational Database Characteristics

- Data in the relational database must be represented in tables, with values in columns within rows.
- Data within a column must be accessible by specifying the table name, the column name, and the value of the primary key of the row.
- The DBMS must support missing and inapplicable information in a systematic way, distinct from regular values and independent of data type.
- The DBMS must support an active on-line catalogue.
- The DBMS must support at least one language that can be used independently and from within programs, and supports data definition operations, data manipulation, constraints and transaction management.
- Views must be updatable by the system.
- The DBMS must support insert, update, and delete operations on sets.

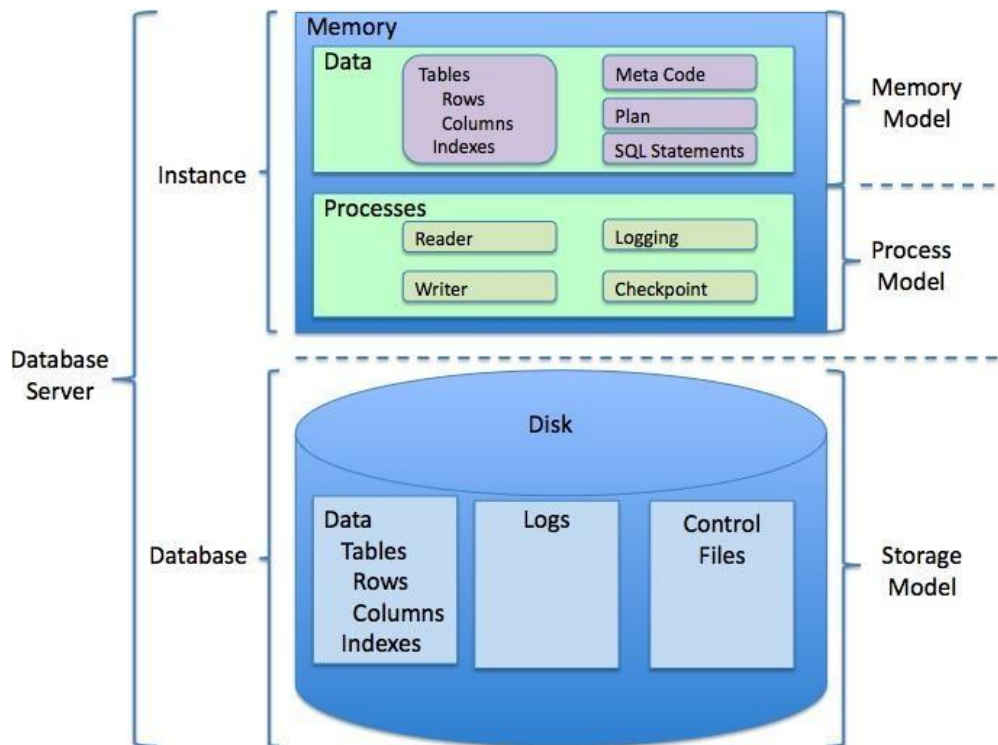


Figure: Relational DBMS (RDBMS)

- The DBMS must support physical and logical data independence.
- Integrity constraints must be stored within the catalogue, separate from the application.
- The DBMS must support distribution independence.
- The existing applications should run when the existing data is redistributed or when the DBMS is redistributed.
- If the DBMS provides a low level interface (row at a time), that interface cannot bypass the integrity constraints.

CODD'S RULE

Explain Codd's rule in detail.

- Dr Edgar F. Codd did some extensive research in Relational Model of database systems and came up with twelve rules of his own which according to him, a database must obey in order to be a true relational database.
- These rules can be applied on a database system that is capable of managing its stored data using only its relational capabilities. This is a foundation rule, which provides a base to imply other rules on it.

Rule Zero

- This rule states that for a system to qualify as an **RDBMS**, it must be able to manage database entirely through the relational capabilities.

Rule 1: Information Rule

- This rule states that all information (data), which is stored in the database, must be a value of some table cell.
- Everything in a database must be stored in table formats. This information can be user data or meta-data.

Rule 2: Guaranteed Access Rule

- This rule states that every single data element (value) is guaranteed to be accessible logically with combination of table-name, primary-key(row value) and attribute-name (column value).
- No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

- This rule states the NULL values in the database must be given a systematic treatment.
- As a NULL may have several meanings, i.e. NULL can be interpreted as one the following: data is missing, data is not known, data is not applicable etc.

Rule 4: Active Online Catalog

- This rule states that the structure description of whole database must be stored in an online catalog, i.e. data dictionary, which can be accessed by the authorized users.
- Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-language Rule

- This rule states that a database must have a support for a language which has linear syntax which is capable of data definition, data manipulation and transaction management operations.
- Database can be accessed by means of this language only, either directly or by means of some application.
- If the database can be accessed or manipulated in some way without any help of this language, it is then a violation.

Rule 6: View Updating Rule

- This rule states that all views of database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-level Insert, Update and Delete Rule

- This rule states the database must employ support high-level insertion, updation and deletion.
- This must not be limited to a single row that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

- This rule states that the application should not have any concern about how the data is physically stored.
- Also, any change in its physical structure must not have any impact on application.

Rule 9: Logical Data Independence

- This rule states that the logical data must be independent of its user's view(application). Any change in logical data must not imply any change in the application using it.
- For example, if two tables are merged or one is split into two different tables, there should be no impact the change on user application. This is one of the most difficult rules to apply.

Rule 10: Integrity Independence

- This rule states that the database must be independent of the application using it.
- All its integrity constraints can be independently modified without the need of any change in the application.
- This rule makes database independent of the front-end application and its interface.

Relational Query Languages

- A **query language** is a language in which a user requests information from the database.
- These languages are usually on a level higher than that of a standard programming language.
- Query languages can be categorized as either procedural or nonprocedural.
 - ✓ In a **procedural language**, the user instructs the system to perform a sequence of operations on the database to compute the desired result.
 - ✓ In a **nonprocedural language**, the user describes the desired information without giving a specific procedure for obtaining that information.
- There are a number of —pure query languages:
 - ✓ The relational algebra is procedural, whereas the tuple relational calculus and domain relational calculus are nonprocedural.

Relational Algebra

Explain the concept of relational algebra in detail. (Or) Explain select, project and Cartesian product operations in relational algebra with an example. (Nov/Dec 2016, April/May 2018)

- The relational algebra is a theoretical procedural query language which takes an instance of relations and does operations on one or more relations to describe another relation without altering the original relation(s).
- The relational algebra defines a set of operations on relations, paralleling the usual algebraic operations such as addition, subtraction or multiplication, which operate on numbers.
- Just as algebraic operations on numbers, the relational algebra consists of a set of operations that take one or two relations as input and produce a new relation as their result.

Operations of Relational Algebra**Unary Operations**

- Select [ρ]
- Project [π]
- Rename [σ]

Binary Operations

- Union [\cup]
- Set Difference [$-$]
- Cartesian Product [\times]

Unary Operations**Select Operation (σ)**

- It selects tuples that satisfy the given predicate from a relation.

Notation – $\sigma_p(r)$

- Where σ stands for selection predicate and r stands for relation. p is propositional logic formula which may use connectors like **and**, **or** and **not**.
- These terms may use relational operators like $=, \neq, \geq, <, >, \leq$.

Example

$\sigma_{\text{subject} = \text{"database"}}(\text{Books})$

Output

- Selects tuples from books where subject is 'database'.

$\sigma_{\text{subject} = \text{"database"} \text{ and } \text{price} = \text{"450"}}(\text{Books})$

Output

- Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{\text{subject} = \text{"database"} \text{ and } \text{price} = \text{"450"} \text{ or } \text{year} > \text{"2010"}}(\text{Books})$

Output

- Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (Π)

- It projects column(s) that satisfy a given predicate.

Notation – $\Pi_{A_1, A_2, \dots, A_n}(r)$

- Where A_1, A_2, \dots, A_n are attribute names of relation r .
- Duplicate rows are automatically eliminated, as relation is a set.

Example

$\Pi_{\text{subject, author}}(\text{Books})$

- Selects and projects columns named as subject and author from the relation Books.

Union Operation (\cup)

- It performs binary union between two given relations and is defined as –

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

Notation – $r \cup s$

- Where r and s are either database relations or relation result set (temporary relation).
- For a union operation to be valid, the following conditions must hold –
 - ✓ r and s must have the same number of attributes.
 - ✓ Attribute domains must be compatible.
 - ✓ Duplicate tuples are automatically eliminated.

$\Pi_{\text{author}}(\text{Books}) \cup \Pi_{\text{author}}(\text{Articles})$

Output

- Projects the names of the authors who have either written a book or an article or both.

Binary Operations***Set Difference (-)***

- The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation – $r - s$

- It finds all the tuples that are present in **r** but not in **s**.

$\Pi_{\text{author}}(\mathbf{Books}) - \Pi_{\text{author}}(\mathbf{Articles})$

Output

- Provides the name of authors who have written books but not articles.

Cartesian Product (X)

- It combines information of two different relations into one.

Notation – $r \times s$

- where **r** and **s** are relations and their output will be defined as $r \times s = \{ q \mid q \in r \text{ and } t \in s \}$

$\sigma_{\text{author} = \text{'tutorialspoint'}}(\mathbf{Books} \times \mathbf{Articles})$

Output

- Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

- The results of relational algebra are also relations but without any name.
- The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** ρ .

Notation – $\rho_x(E)$

- Where the result of expression **E** is saved with name of **x**. Additional

operations are –

- Set Intersection
- Assignment
- Natural join

Relational Calculus

- In contrast to Relational Algebra, Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it.
- Relational calculus exists in two forms –
 - ✓ Tuple Relational Calculus (TRC)
 - ✓ Domain Relational Calculus (DRC)

Tuple Relational Calculus (TRC)

- Filtering variable ranges over tuples

Notation – $\{T \mid \text{Condition}\}$

- Returns all tuples **T** that satisfies a condition.

Example

$$\{ T.name \mid \text{Author}(T) \text{ AND } T.article = 'database' \}$$
Output

- Returnstuples with'name' from Author whohaswrittenarticle on'database'.
- TRC can be quantified. We can use Existential(\exists) and Universal Quantifiers(\forall).

Example

$$\{ R \mid \exists T \in \text{Authors}(T.article='database' \text{ AND } R.name=T.name) \}$$
Output

- The above querywill yield the sameresult as the previous one.

Domain Relational Calculus (DRC)

- InDRC,thefilteringvariableusesthedomainofattributesinsteadofentiretuplevalues (as done in TRC, mentioned above).

Notation – $\{ a_1, a_2, a_3, \dots, a_n \mid P(a_1, a_2, a_3, \dots, a_n) \}$

- Where a_1, a_2 are attributes and P stands for formulae built by inner attributes.

Example

$$\{ \langle article, page, subject \rangle \mid \in \text{TutorialsPoint} \wedge \text{subject} = 'database' \}$$
Output

- Yields Article, Page, and Subject from the relation TutorialsPoint, where subject is database.
- Just like TRC, DRC can also be written using existential and universal quantifiers. DRC also involves relational operators.
- The expression power of Tuple Relation Calculus and Domain Relation Calculus is equivalent to Relational Algebra.

SQL Fundamentals**What is SQL? (Or) Explain detail about the fundamentals of SQL.**

- SQL stands for Structured Query Language.
- SQL is a standard language for accessing and manipulating databases.
- The tasks related to relational data management— creating tables, querying the database for information, modifying the data in the database, deleting them, granting access to users, and so on.

(Or)

- SQL stands for Structured Query Language.
- It is a programming language which stores, manipulates and retrieves the stored data in RDBMS.
- SQL syntax is not case sensitive.
- SQL is standardized by both ANSI and ISO.
- It is a standard language for accessing and manipulating databases.

Characteristics of SQL

- SQL is extremely flexible.
- SQL uses a free form syntax that gives the ability to user to structure the SQL statements in a best suited way.
- It is a high level language.
- It receives natural extensions to its functional capabilities.
- It can execute queries against the database.

Advantages of SQL

- SQL provides a greater degree of abstraction than procedural language.
- It is coded without embedded data-navigational instructions.
- It enables the end users to deal with a number of database management systems where it is available.
- It retrieves quickly and efficiently huge amount of records from a database.
- No coding required while using standard SQL.

Roles of SQL

- SQL retrieves data from the database. It is **an interactive query language**.
- It can be used along with programming language to access data from database. It is **a database programming language**.
- It can be used to monitor and control data access by various users. It is **a database administration language**.
- It can be used as **an Internet data access language**.

SQL Datatypes

The SQL standard supports a variety of built-in domain types, including:

- char(*n*)**: A fixed-length character string with user-specified length *n*. The full form, **character** can be used instead.
- varchar(*n*)**: A variable-length character string with user-specified maximum length *n*. The full form, **character varying**, is equivalent.
- int**: An integer (a finite subset of the integers that is machine dependent). The full form, **integer**, is equivalent.
- smallint**: A small integer (a machine-dependent subset of the integer domain type).
- numeric(*p,d*)**:
 - ✓ A fixed-point number with user-specified precision.
 - ✓ The number consists of *p* digits (plus a sign), and *d* of the *p* digits are to the right of the decimal point.
 - ✓ Thus, **numeric(3,1)** allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.
- real, double precision**: Floating-point and double precision floating-point numbers with machine-dependent precision.
- float(*n*)**: A floating-point number, with precision of at least *n* digits.

- ❑ **date**: a calendar date containing a (four-digit) year, month, and day of the month.
- ❑ **time**:
 - ✓ The time of day, in hours, minutes and seconds.
 - ✓ A variant, **time**(*p*), can be used to specify the number of fractional digits for seconds (the default being 0).
 - ✓ It is also possible to store time zone information along with the time.
- ❑ **timestamp**: A combination of **date** and **time**. A variant, **timestamp**(*p*), can be used to specify the number of fractional digits for seconds (the default here being 6).
- ❑ Date and time values can be specified like this:
 - date** `'2001-01-24'`
 - time** `'09:30:00'`
 - timestamp** `'2001-04-25 10:29:01.45'`
- ❑ Dates must be specified in the format year followed by month followed by day, as shown.

SQL Languages

State and explain the command DDL, DML, DCL with suitable example. (Nov/Dec 2017)

SQL Command Types

- ❑ SQL commands can be divided into two main sub-languages.
- ❑ **Data Definition Language (DDL)**
 - ✓ It contains the commands used to create and destroy databases and database objects.
- ❑ **Data Manipulation Language (DML)**
 - ✓ After the database structure is defined with DDL, database administrators and users can use the DML commands.
 - ✓ It is used to insert, retrieve and modify the data contained within it.
- ❑ **Data Control Language (DCL)**
 - ✓ It is used to control the access privilege to the database.
 - ✓ DCL provides two commands such as grant and revoke.
- ❑ **Transaction Control Language (TCL)**
 - ✓ It is used to control and manage transactions to maintain the integrity of data within SQL statements.
 - ✓ TCL provides command such as commit, rollback, etc.
- ❑ **View Definition**: The SQL DDL includes commands for defining views.

DDL Commands

- ❑ The Data Definition Language is used to create and destroy databases and database objects.
- ❑ These commands are primarily used by database administrators during the setup and removal phases of a database project.
- ❑ The four basic DDL commands are,

Create Command (Database)

- ❑ It allows you to create and manage many independent databases.

Syntax

Create database <database name>

Example

Create database employee

Create Command (Table)

- It is used to create a table.

Syntax

create table <table name> (columnname1 datatype(size), columnname2 datatype(size)...);

Example

SQL>**create table** employee (ename **varchar**(10), eid **number**(5), address **varchar2**(10), salary **number**(5), designation varchar2(10));

Use Command

- It allows you to specify the database you want to work with within your DBMS.

Syntax

Use <database name>

Example

Use Employee

Alter Command

- It is used to add a new column or modify existing column definitions.

Syntax

alter table <tablename> **add** (new columnname1 datatype(size), new columnname2 datatype(size)...);

alter table <table name> **modify** (column definition);

Example

SQL>**alter table** employee **modify**(eid **number**(7));

SQL>**alter table** employee **add** (age **number**(2));

Drop Command

- It is used to delete a table.

Syntax

drop table <tablename>;

Example

SQL>**drop table** employee;

Notes: This command will delete the contents as well as structure.

Truncate Command

- It is used to delete the records but retain the structure.

Syntax

truncate table <tablename>;

Example

SQL>**truncate table** employee;

To view the table structure**Syntax**

desc <tablename>;

Example

SQL>**desc** employee;

DML Commands

- It is used to retrieve, insert and modify database information.
- These commands are used by all database users during the routine operation of the database.

Insert Command

- It is used to insert a new record in the database.

Syntax

insert into <tablename> **values** (a list of data values);

Example

SQL>**create table** employee (ename varchar2(10), eid number (5), salary number(5));

SQL>**insert into** employee **values**(‘_ABC’,50,1000); **Select**

Command

- The SELECT command is the most commonly used command in SQL.
- It allows database users to retrieve the specific information they desire from an operational database.

Syntax

Select * from <table name>

Example

Select * from employee

Update Command

- It is used to modify (update) the information contained within a table, either in bulk or individually.

Syntax

update <tablename> **set** field=value,..... **where** <condition>;

Example

SQL>**update** employee **set** eid=100 **where** ename = ‘_ABC’;

Delete Command

Rows can be deleted using delete command

Syntax

delete from <tablename> **where** <condition>;

Example

SQL>**delete** from employee **where** eid=100;

DCL Commands

- It is used to control privilege in Database.
- To perform any operation in the database, such as for creating tables, sequences or views we need privileges.

Grant Command

- It gives user access privileges to database.

Syntax

Grant Select/insert/delete/update/alter/all privileges on tablename to authenticate;

Example

Grant select, update on student to vikram;

Revoke Command

- It takes back permissions from user.

Syntax

Revoke Select/insert/delete/update/alter/all privileges on tablename from authenticate;

Example

Revoke select, update on student from vikram;

TCL Commands

- Used to manage transactions in database
- To manage the changes made by DML statements.
- Allows statements to be grouped together into logical transactions.

Commit command

- Commit command is used to permanently save any transaction into database.

Syntax

Commit;

Rollback Command

- It restores the database to last committed state.
- It is also use with savepoint command to jump to a savepoint in a transaction.

Syntax

rollback to savepoint_name;

Example

rollback to Temp;

Savepoint Command

- It is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

Syntax

Savepoint savepoint_name;

Example

Savepoint Temp

Some of the Most Important SQL Commands

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database

- ❑ **ALTER DATABASE** - modifies a database
- ❑ **CREATE TABLE** - creates a new table
- ❑ **ALTER TABLE** - modifies a table
- ❑ **DROP TABLE** - deletes a table
- ❑ **CREATE INDEX** - creates an index (search key)
- ❑ **DROP INDEX** - deletes an index

Advanced SQL Features

Write short notes on advanced SQL features.

- ❑ The structure of an SQL expression consists of three clauses: select, from and where
 - ✓ The **select clause** corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query.
 - ✓ The **from clause** corresponds to the Cartesian product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.
 - ✓ The **where clause** corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the from clause.
- ASQL has the form,

select A₁, A₂, A_n
from r₁, r₂, r_m
where p

A_i - an attribute r_i -
 relation
 p - predicate
- The query is equivalent to the relational algebra expression. A₁, A₂, A_n
 ($\sigma_p(r_1 \times r_2 \times \dots \times r_m)$)
- ❑ If the where clause is omitted, the predicate p is true.

Tuple Variables

- ❑ **Tuple variables** are defined in the from clause via the use of the **as clause**.
- ❑ For example, find the customer names and their loan numbers for all customers having a loan at some branch in the banking database.
 SQL > **select** B.customer_name, B.loan_no, L.amount **from** borrower as B, loan as L **where**
 L.loan_no = B.loan_no;

String Operations

- ❑ SQL includes a string matching operator for comparisons on character strings. Patterns are described using 2 special characters.
 - ✓ **Percent (%):** The % character, matches any substring.
 - ✓ **Underscore (-):** The - character matches any character.

Example:

Find the names of customers where the 1st 2 characters are `_Ba'`.

SQL>**select** customer_name **from** customer **where** customer_name **like** `_Ba%`; Find the names of customer where the 2nd character is `_n'` or `_a'`.

SQL>**select** customer_name **from** customer **where** customer_name **like** `_n%` or `_a%`; Patterns are case sensitive, i.e., uppercase characters do not match lowercase characters, and vice versa.

SQL supports a variety of string operations such as

- ✓ Concatenation using `||` or `strcat ()`
- ✓ Converting the string into upper or lower case `upper ()` or `lower ()`
- ✓ Find string length (`strlen()`), extracting substring (`substr ()`),
etc.

Order by Clause

- The order by clause causes the tuples in the result of a query to appear in sorted order.

SQL>**select** *

from employee

order by salary;

SQL>**select** *

from employee

order by salary **desc**, eid **asc**;

Aggregate Functions**Explain the aggregate functions in SQL with an example. (April/May 2018)**

- Aggregate functions are functions that take a collection of values as input and return a single value as output.

SQL offers 5 built-in aggregate functions: avg

- average value
- min - minimum value
- max - maximum value
- sum - sum of values
- count - number of values.

Examples

1. Find the average account balance at the perryridge branch:

SQL>**select avg** (balance) **from** account **where**

branch-name = 'Perryridge';

2. Find the number of tuples in the customer relation

SQL>**select count** (*) **from** customer;

Aggregate functions with group by clause

- Group by clause is used to group the rows based on certain criteria. Group by is used in conjunction with aggregate functions like sum, avg, min, max, count, etc.

Example: Find the average account balance at each branch.

SQL>**select** branch_name, **avg** (balance) **from** account **group by** branch_name;

Aggregate functions with having clause

- Find the name of all branches where the average account balance is more than \$2,000. SQL>**select** branch_name, **avg** (balance) **from** account **group by** branch_name **having avg** (balance) > 2000;

Null Values

- SQL allows the use of null values to indicate absence of information about the value of an attribute.
- Null signifies an unknown value or that a value does not exist.
- The predicate is null can be used to check for null values.

Example:

- Find all loan numbers which appear in the loan relation with null values for amount. SQL>**select** loan_no **from** loan **where** amount **is null**;
- The result of any arithmetic expression is null if any of the input values is null.

Example:

5 + null returns null.

Consider the unknown (null) value used in boolean expressions as,

- OR - (unknown or true) = true, (unknown or false) = unknown, (unknown or unknown) = unknown.
- AND - (true and unknown) = unknown, (false and unknown) = false, (unknown and unknown) = unknown.
- NOT - (not unknown) = unknown. For

example, find total of all loan amounts.

SQL>**select sum** (amount) **from** loan;

Above statement ignores all null amounts. The result is null if there is no non- null amount.

- All aggregate operations except count (*) ignore tuples with null values on the aggregated attributes.

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A subquery is a select-from-where expression that is nested within another query.
- A common use of subquery is to perform tests for set membership, set comparisons and set cardinality.

Set Membership

- SQL uses in and not in constructs to set membership tests.

In

- The **in** connective tests for set membership, where the set is a collection of values produced by a select clause.

Example: Find all customers who have both an account and a loan at the bank.

SQL>**select distinct** customer-name **from** depositor
where customer-name **in** (**select** customer-name **from** borrower);

Not In

- The **not in** connective tests for the absence of set membership.

Example: Find all customers who have a loan at the bank but do not have an account at the bank.

SQL>**select distinct** customer-name **from** borrower
where customer-name **not in** (**select** customer name **from** depositor);

Set Comparison

- Nested queries are used to compare sets. SQL uses various comparison operators such as <, >, <=, >=, <>, any, all, and, some, etc. to compare sets.

Example:

Find the names of all branches that have assets greater than those of at least one branch located in —Chennai.

SQL>**select distinct** T.branchname **from** branches T, branch
as S **where** T. assets > S. assets
and S. branch-city = —Chennai; Same

query using > **some** clause.

SQL>**select** branch-name **from** branch **where** assets > **some** (**select** assets **from** branch
where branch-city = —Chennai);

- SQL also allows < **some**, <= **some**, >= **some**, = **some**, and <> **some** comparisons, = **some** is identical to in, and <> **some** is identical to not in. The keyword **any** is synonym to **some** in SQL.
- SQL also allows < **all**, > **all**, <= **all**, >= **all**, = **all**, and <> **all** comparisons <> **all** is identical to **not in**.

Example:

Find the names all branches that have an-assets value greater than that of each branch in —Chennai.

SQL>**select** branch-name **from** branch **where** assets > **all** (**select** assets **from** branch **where**
branch-city = —Chennai);

Test for Empty Relations

- SQL includes a feature for testing whether a subquery has any tuples in its result.
- The **exists** construct returns the value true if the arguments subquery is non-empty.

Example:

Find all customers who have both an account and a loan at the bank.

SQL>**select** customer-name **from** borrower **where** exists (**select** * **from** depositor **where**
depositor.customer-name = borrower.customer-name);

- Similar to **exists** we can use **not exists** also. Find all customers who have an account at all branches located in —Chennai.

SQL>**select distinct** S. customer-name **from** depositor **as** S
where not exists ((**select** branch-name **from** branch

II Year / IV Semester - CSE

where branch-city = —Chennai)**except**
 (**select** R. branch_name **from** depositor **as** T, account **as** R
where T. account_no = R. account_no **and**
 S. customer-name = T. customer-name));

Test for absence of duplicate tuples

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.

Example:

Find all customers who have at most one account at the —Chennai branch.

```
SQL>select T. customer_name from depositor as T
where unique (select R.customer-name
from account, depositor as R
where T.customer_name=R.customer-name and R.
account-no=account.account_no and
account.branch_name=—Chennai);
```

- **not unique** construct is used to test the existence of duplicate tuples in the same manner.

Complex Queries

- Complex queries are often hard or impossible to write as a single SQL block.
- There are two ways for composing multiple SQL blocks to express a complex query.
 - ✓ Derived Relations
 - ✓ With Clause

Derived Relations

- SQL allows a subquery expression to be used in the **from** clause.
- If we use such an expression, then we must give the result relation a name, and we can rename the attributes. For renaming **as** clause is used.
- For example, find the average account balance of those branches where the average account balance is greater than \$ 2000.

```
SQL>select branch-name, avg-balance from (select branch-name,
avg (balance)from account group by branch-name) as
branch-avg (branch-name, avg-balance) where avg-balance
>2000;
```

- Here subquery result is named branch-avg with the attributes branch-name and avg- balance.

With Clause

- The with clause provides a way of defining a temporary view, whose definition available only to the query in which the **with** clause occurs.
- Consider the following query, which selects **accounts with the maximum balance**.
- If there are many accounts with the same maximum balance, all of them are selected. with max_balance (value) as


```
select max (balance)
```

```

from account
select account_no from account, max_balance where
account.balance = max_balance.value;

```

Views

Write short notes on views.

- A view is an object that gives the user a logical view of data from an underlying table or tables (relation or relations).
- It is not desirable for all users to see the entire logical model.
- Security considerations may require that certain data be hidden from users.
- Any relation that is not part of the logical model, but is made visible to a user as a **virtual relation**, is called as **view**.
- Views may be created for the following reasons:
 - ✓ To provide Data Security
 - ✓ Query Simplicity
 - ✓ Structural Simplicity (because view contains only limited number of columns and rows).

(1) Creation of Views

Syntax

```
create view view_name as < query expression >;
```

Example:

```

Create a view customer_details from customer relation with customer name and customer-id.
create view customer_details
as
select customer_name, customer_id from customer;

```

(2) Assigning Names to Columns

- We can assign names for the various columns in the view.
- This may be entirely different from what has been used in the main relation. For example,


```
SQL>create view customer_details (cust_name, customer_no)
as select customer_name, customer_id from customer;
```

(3) Selecting data from a view

```
SQL>select * from customer_details;
```

(4) Updation of a view

- Views can also be used for data manipulation i.e., the user can perform insert, update, and the delete operations on the view.
- The views on which data manipulation can be done are called **Updatable views**, the views that do not allow data manipulation are called **Readonly views**.
- When you give a view name in the update, insert, or delete statement, the modification to the data will be passed to the underlying (main) relation.

For the view to be updatable, it should meet following criteria:

- The view must be created on a single table.
- Primarykeycolumn of the table should beincluded inthe view.
- Aggregate functions cannot be used in the select statement.
- The select statement used for creating a view should not include distinct, group by or having clause.
- Select statement used for creatinga viewshould notincludesubqueries.
- It must not use constant, strings or value expression like total/6.

(5) Destroying a View

A view can be dropped by using the drop view command.

Syntax

drop view view_name;

Example

SQL>**drop view** customer_details;

Joins**Explain in detail about join operation in SQL.**

- Ajoinisaqueryusingwhich we canquerydatamore thanonetable.
- Joins are the basic of multi-table query processing in SQL.
- Ajoinisaquerythatextractscorrespondingrowsfromtwoormorettables, viewsor snapshots.
- If the two tables used in the join have the same column name, then the column names should be prefixed with table name followed by a period.
- SELECT statement of a multi-table query must contain a filter condition that specify the column match. The **where** clause is used to specify the selection condition and the join condition. In the where clause the logical operators can also be used.

Types of Joins

Joins are classified into four types namely:

- Inner Join
- Outer Join
- Natural Join

Inner Join

- Inner joinreturns the matchingrows from the tables that arebeingjoined.
- Consider followingtwo relations:
 - ✓ Student(sname, place)
 - ✓ Student_marks(sname, dept, mark)

II Year / IV Semester - CSE

Student	sname	place
	Prajan	Chennai
	Anand	Kolkata
	Kumar	Delhi
	Ravi	Mumbai

Student_marks	sname	dept	mark
	Prajan	CS	700
	Anand	IT	650
	Vasu	CS	680
	Ravi	IT	600

Example 1

SQL>**select** Student.sname, Student_marks, mark **from** Student **inner join** Student_marks **on** Student.sname=Student_marks.sname; The output of the above query is,

sname	mark
Prajan	700
Anand	650
Ravi	600

Example 2

SQL>**select** * **from** Student **inner join** Student_marks **on** Student.sname=Student_marks.sname; The result of the above query is,

sname	place	sname	dept	mark
Prajan	Chennai	Prajan	CSE	700
Anand	Kolkata	Anand	IT	650
Ravi	Mumbai	Ravi	IT	600

- For example 2 the result consists of the attributes of the left-hand-side relation followed by the attributes of the right-hand-side relation.
- Thus, the sname attribute appears twice in result, first is from student table and second is from student_marks table.

Outer Join

- Whentablesarejoinedusinginnerjoin, rowswhichcontainmatchingvaluesinthejoin predicate are returned.
- Sometimesyoumaywantbothmatchingandnon-matchingrowsreturnedforthetables thatarebeingjoined. Thiskind ofoperation isknown as an outerjoin.
- An outer join is an extended form of the inner join.

- In this, the rows in one table having no matching rows in the other table will also appear in the result table with nulls.

Types of Outer Join

The Outer Join can be any one of the following:

- ✓ Left Outer
- ✓ Right Outer

1. Left Outer join

- The left outer join returns matching rows from the tables being joined and also non-matching rows from the left table in the result and places null values in the attributes that come from the right table.

Example 3

SQL> **select** Student.sname, Student_marks.mark **from** Student **left outer join** Student_marks **on** Student.sname = Student_marks.sname; The result of

above query is

sname	mark
Prajan	700
Anand	650
Ravi	600
Kumar	null

Left outer join operation is computed as follows:

- First compute the result of inner join as before.
- Then, for every tuple $_t'$ in the left hand side relation, Student that does not match any tuple in the right-hand side relation Student_marks in the inner join, add a tuple $_r'$ to the result of the join:
- The attributes of tuple $_r'$ that are derived from the left-hand side relation are filled with from tuple $_t'$, remaining attributes of $_r'$ are filled with null values as shown in example 3.

2. Right outer join

- The right outer join operation returns matching rows from the tables being joined, and also non-matching rows from the right table in the result and places null values in the attributes that comes from the left table.

Example 4

SQL> **select** Student.sname, Student.place, Student_marks.mark **from** Student **right outer join** Student_marks **on** Student.sname = Student_marks.sname;

The result of above query is,

sname	place	mark
Prajan	Chennai	700
Anand	Kolkota	650
Ravi	Mumbai	600
Vasu	null	680

Embedded SQL

Explain in detail about Embedded SQL.

- Embedded SQL is a method of combining the computing power of a programming language and the database manipulation capabilities of SQL.
- A language in which SQL queries are embedded is referred to as a *host* language, and the SQL structures permitted in the host language constitute *embedded SQL*.
- Programs written in the host language can use the embedded SQL syntax to access and update data stored in a database.
- An embedded SQL program must be processed by a special preprocessor (SQL Preprocessor) prior to compilation.
- The preprocessor replaces embedded SQL requests with host-language declarations and procedure calls that allow runtime execution of the database accesses.
- The output from the preprocessor is then compiled by the host language compiler.
- This allows programmers to embed SQL statements in programs written in any number of languages such as C/C++, Java, COBOL and FORTRAN.
- To identify embedded SQL requests to the preprocessor, we use the EXEC SQL statement; it has the form:

EXEC SQL <embedded SQL statement >;

- The exact syntax for embedded SQL requests depends on the language in which SQL is embedded.

Embedded SQL in C Program Examples

Example 1

/* Variable Declaration in Language C */

- Variables inside **DECLARE** are shared and can appear (while prefixed by a colon) in SQL statements
- **SQLCODE** is used to communicate errors/exceptions between the database and the program

```
int loop;
EXEC SQL BEGIN DECLARE SECTION;
    varchar dname[16], fname[16], ...; char
    ssn[10], bdate[11], ...;
    int dno, dnumber, SQLCODE, ...; EXEC
SQL END DECLARE SECTION;
```

Example 2

```
/* Conditional and Looping Statements in Language C */
```

```
loop = 1; while
(loop) {
    prompt(—EnterSSN:—,ssn);
    EXEC SQL
        select FNAME, LNAME, ADDRESS, SALARY into :fname, :lname, :address,
        :salary from EMPLOYEE where SSN == :ssn; if
        (SQLCODE == 0) printf(fname, ...);
        else printf(—SSN does not exist: —, ssn);
        prompt(—MoreSSN?(1=yes,0=no):—,loop);
    END-EXEC
}
```

Dynamic SQL

- **Dynamic SQL** is a programming methodology for generating and running SQL statements at runtime.
- It is useful when writing general-purpose and flexible programs like dynamic query systems, when writing programs that must run database definition language (DDL) statements, or when you do not know at compile time the full text of a SQL statement or the number or data types of its input and output variables.

Difference between Static SQL and Dynamic SQL

S. No.	Static SQL	Dynamic SQL
1.	In static SQL how database will be accessed is predetermined in the embedded SQL statement.	In dynamic SQL, how database will be accessed is determined at runtime.
2.	It is less flexible and more efficient.	It is more flexible and less efficient.
3.	SQL statements are compiled at compile time.	SQL statements are recompiled at runtime.
4.	Parsing, validation, optimization, and generation of application plan are done at compile time.	Parsing, validation, optimization, and generation of application plan are done at run time.
5.	It is generally used for situations where data is distributed uniformly.	It is generally used for situations where data is distributed non-uniformly.
6.	EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are not used.	EXECUTE IMMEDIATE, EXECUTE and PREPARE statements are used.



MADHA
Expertise | Empathy | Excellence
ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COMMON FOR: DEPARTMENT OF INFORMATION TECHNOLOGY

CS8493- Operating System

R – 2017

LECTURE NOTES

CS8493- Operating System

UNIT-I

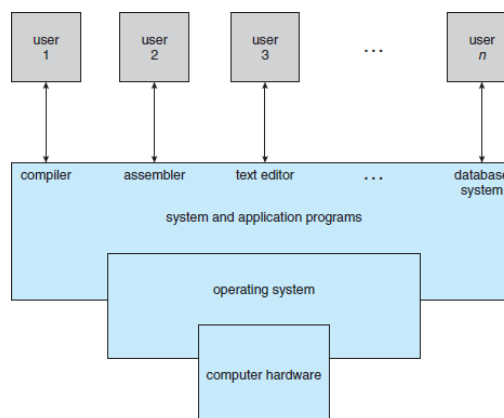
1.1 introduction

An *operating system* acts as an intermediary between the user of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user can execute programs in a *convenient* and *efficient* manner.

An **operating system** is a program that manages a computer's hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware. An amazing aspect of operating systems is how they vary in accomplishing these tasks. Mainframe operating systems are designed primarily to optimize utilization of hardware. Personal computer (PC) operating systems support complex games, business applications, and everything in between. Operating systems for mobile computers provide an environment in which a user can easily interface with the computer to execute programs. Thus, some operating systems are designed to be *convenient*, others to be *efficient*, and others to be some combination of the two.

1.2 Computer system overview

A computer system can be divided roughly into four components: the *hardware*, the *operating system*, the *application programs*, and the *users* (Figure 1.1). The **hardware**—the **central processing unit (CPU)**, the **memory**, and the **input/output (I/O) devices**—provides the basic computing resources for the system. The **application programs**—such as word processors, spreadsheets, compilers, and Web browsers—define the ways in which these resources are used to solve users' computing problems. The operating system controls the hardware and coordinates its use among the various application programs for the various users.



1.1.1 User View

The user's view of the computer varies according to the interface being used. Most computer users sit

in front of a PC, consisting of a monitor, keyboard, mouse, and system unit. Such a system is designed for one user to monopolize its resources. The goal is to maximize the work (or play) that the user is performing. In this case, the operating system is designed mostly for **ease of use**, with some attention paid to performance and none paid to **resource utilization**—how various hardware and software resources are shared.

Performance is, of course, important to the user; but such systems are optimized for the single-user experience rather than the requirements of multiple users.

1.1.2 System View

From the computer's point of view, the operating system is the program most intimately involved with the hardware. In this context, we can view an operating system as a **resource allocator**. A computer system has many resources that may be required to solve a problem: CPU time, memory space, file-storage space, I/O devices, and so on. The operating system acts as the manager of these resources. Facing numerous and possibly conflicting requests for resources, the operating system must decide how to allocate them to specific programs and users so that it can operate the computer system efficiently and fairly. As we have seen, resource allocation is especially important where many users access the same mainframe or minicomputer.

1.3 Basic elements

A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory (Figure 1.2). Each device controller is in charge of a specific type of device (for example, disk drives, audio devices, or video displays). The CPU and the device controllers can execute in parallel, competing for memory cycles. To ensure orderly access to the shared memory, a memory controller synchronizes access to the memory.

For a computer to start running—for instance, when it is powered up or rebooted—it needs to have an initial program to run. This initial program, or **bootstrap program**, tends to be simple. Typically, it is stored within the computer hardware in read-only memory (**ROM**) or electrically erasable programmable read-only memory (**EEPROM**), known by the general term **firmware**. It initializes all aspects of the system, from CPU registers to device controllers to memory contents. The bootstrap program must know how to load the operating system and how to start executing that system. To accomplish this goal, the bootstrap program must locate the operating-system kernel and load it into memory. Once the kernel is loaded and executing, it can start providing services to the system and its users. Some services are provided outside of the kernel, by system programs that are loaded into memory at boot time to become **system processes**, or **system daemons** that run the entire time the kernel is running.

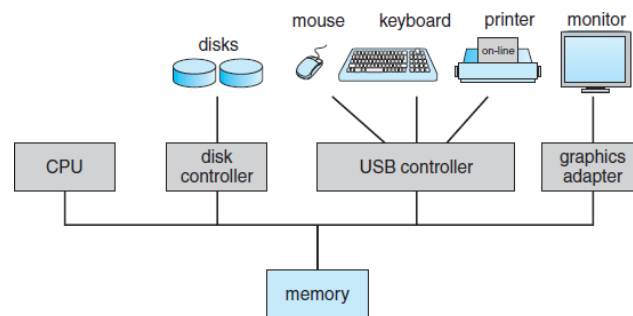


Figure 1.2 A modern computer system.

1.4 Instruction Execution and Interrupts

The occurrence of an event is usually signaled by an **interrupt** from either the hardware or the software. Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the

system bus. Software may trigger an interrupt by executing a special operation called a **system call** (also called a **monitor call**).

When the CPU is interrupted, it stops what it is doing and immediately transfers execution to a fixed location. The fixed location usually contains the starting address where the service routine for the interrupt is located. The interrupt service routine executes; on completion, the CPU resumes the Interrupted computation. A timeline of this operation is shown in Figure 1.3.

Interrupts are an important part of computer architecture. Each computer design has its own interrupt mechanism, but several functions are common. The interrupt must transfer control to the appropriate interrupt service routine. The straightforward method for handling this transfer would be to invoke a generic routine to examine the interrupt information. The routine, in turn, would call the interrupt-specific handler. However, interrupts must be handled quickly. Since only a predefined number of interrupts is possible, a table of pointers to interrupt routines can be used instead to provide the necessary speed. The interrupt routine is called indirectly through the table, with no intermediate routine needed. Generally, the table of pointers is stored in low memory (the first hundred or so locations). These locations hold the addresses of the interrupt service routines for the various devices. This array, or **interrupt vector**, of addresses is then indexed by a unique device number, given with the interrupt request, to provide the address of the interrupt service routine for the interrupting device.

Chapter 1 Introduction

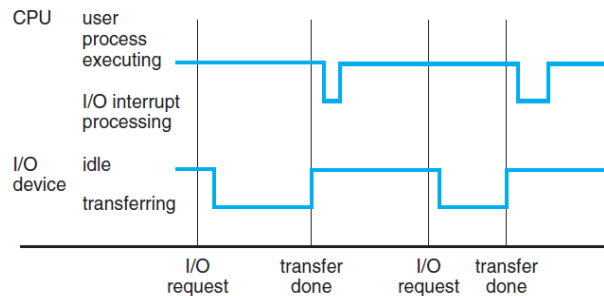


Figure 1.3 Interrupt timeline for a single process doing output.

1.5 Memory Hierarchy

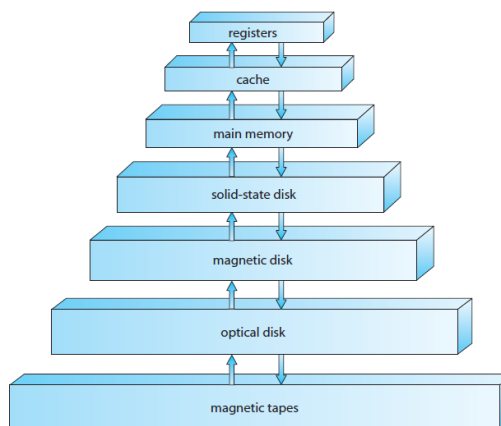


Figure 1.4 Storage-device hierarchy.

The wide variety of storage systems can be organized in a hierarchy (Figure 1.4) according to speed and cost. The higher levels are expensive, but they are fast. As we move down the hierarchy, the cost per bit generally decreases, whereas the access time generally increases. This trade-off is reasonable; if a given storage system were both faster and less expensive than another—other properties being the same—then there would be no reason to use the slower, more expensive memory. In fact, many early storage devices, including paper tape and core memories, are relegated to museums now that magnetic tape and **semiconductor memory** have become faster and cheaper. The top four levels of memory in Figure 1.4 may be constructed using semiconductor memory.

In addition to differing in speed and cost, the various storage systems are either volatile or nonvolatile. As mentioned earlier, **volatile storage** loses its contents when the power to the device is removed. In the absence of expensive battery and generator backup systems, data must be written to **nonvolatile storage** for safekeeping. In the hierarchy shown in Figure 1.4, the storage systems above the solid-state disk are volatile, whereas those including the solid-state disk and below are nonvolatile.

Solid-state disks have several variants but in general are faster than magnetic disks and are nonvolatile. One type of solid-state disk stores data in a large DRAM array during normal operation but also contains a hidden magnetic hard disk and a battery for backup power. If external power is interrupted, this solid-state disk's controller copies the data from RAM to the magnetic disk. When external power is restored, the controller copies the data back into RAM. Another form of solid-state disk is flash memory, which is popular in cameras and **personal digital assistants (PDAs)**, in robots, and increasingly for storage on general-purpose computers. Flash memory is slower than DRAM but needs no power to retain its contents. Another form of nonvolatile storage is **NVRAM**, which is DRAM with battery backup power. This memory can be as fast as DRAM and (as long as the battery lasts) is nonvolatile.

1.6 Cache Memory

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
 - ⊙ If it is, information used directly from the cache (fast)
 - ⊙ If not, data copied to cache and used there
- Cache smaller than storage being cached
 - ⊙ Cache management important design problem
 - ⊙ Cache size and replacement policy

1.7 Direct Memory Access

Interrupt-driven I/O is fine for moving small amounts of data but can produce high overhead when used for bulk data movement such as disk I/O. To solve this problem, **direct memory access (DMA)** is used. After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU. Only one interrupt is generated per block, to tell the device driver that the operation has completed, rather than the one interrupt per byte generated for low-speed devices. While the device controller is performing these operations, the CPU is available to accomplish other work.

Some high-end systems use switch rather than bus architecture. On these systems, multiple components can talk to other components concurrently, rather than competing for cycles on a shared bus. In this case, DMA is even more effective. Figure 1.5 shows the interplay of all components of a computer system.

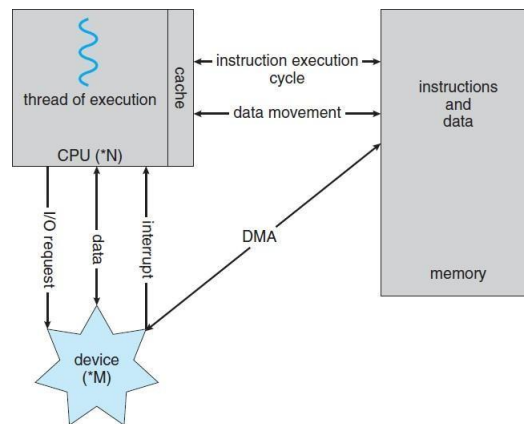


Figure 1.5 How a modern computer system works.

1.7 Multiprocessor and Multicore Organization

1.7.1 Multiprocessor Organization

Within the past several years, **multiprocessor systems** (also known as **parallel systems** or **multicore systems**) have begun to dominate the landscape of computing. Such systems have two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices. Multiprocessor systems first appeared prominently in servers and have since migrated to desktop and laptop systems. Recently, multiple processors have appeared on mobile devices such as smart phones and tablet computers. Multiprocessor systems have three main advantages:

1. Increased throughput. By increasing the number of processors, we expect to get more work done in less time. The speed-up ratio with N processors is not N , however; rather, it is less than N . When multiple processors cooperate on a task, a certain amount of overhead is incurred in keeping all the parts working correctly. This overhead, plus contention for shared resources, lowers the expected gain from additional processors. Similarly, N programmers working closely together do not produce N times the amount of work a single programmer would produce.

2. Economy of scale. Multiprocessor systems can cost less than equivalent multiple single-processor systems, because they can share peripherals, mass storage, and power supplies. If several programs operate on the same set of data, it is cheaper to store those data on one disk and to have all the processors share them than to have many computers with local disks and many copies of the data.

3. Increased reliability. If functions can be distributed properly among several processors, then the failure of one processor will not halt the system, only slow it down. If we have ten processors and one fails, then each of the remaining nine processors can pick up a share of the work of the failed processor. Thus, the entire system runs only 10 percent slower, rather than failing altogether.

The multiple-processor systems in use today are of two types. Some systems use **asymmetric multiprocessing**, in which each processor is assigned a specific task. A **boss** processor controls the system; the other processors either look to the boss for instruction or have predefined tasks. This scheme defines a boss–

worker relationship. The boss processor schedules and allocates work to the worker processors. The most common systems use **symmetric multiprocessing (SMP)**, in which each processor performs all tasks within the operating system. SMP means that all processors are peers; no boss–worker relationship exists between processors. Figure 1.6 illustrates a typical SMP architecture.

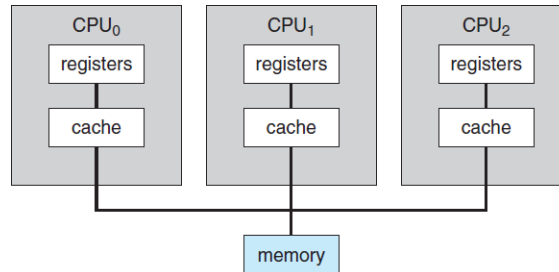


Figure 1.6 Symmetric multiprocessing architecture.

Multiprocessing adds CPUs to increase computing power. If the CPU has an integrated memory controller, then adding CPUs can also increase the amount of memory addressable in the system. Either way, multiprocessing can cause a system to change its memory access model from uniform memory access (**UMA**) to non-uniform memory access (**NUMA**). UMA is defined as the situation in which access to any RAM from any CPU takes the same amount of time. With NUMA, some parts of memory may take longer to access than other parts, creating a performance penalty. Operating systems can minimize the NUMA penalty through resource management

1.7.2 Multicore Organization

Earlier in the history of computer design, in response to the need for more computing performance, single-CPU systems evolved into multi-CPU systems. A more recent, similar trend in system design is to place multiple computing cores on a single chip. Each core appears as a separate processor to the operating system (Section 1.3.2). Whether the cores appear across CPU chips or within CPU chips, we call these systems **multicore** or **multiprocessor** systems. Multithreaded programming provides a mechanism for more efficient use of these multiple computing cores and improved concurrency. Consider an application with four threads. On a system with a single computing core, concurrency merely means that the execution of the threads will be interleaved over time (Figure 4.3), because the processing core is capable of executing only one thread at a time. On a system with multiple cores, however, concurrency means that the threads can run in parallel, because the system can assign a separate thread to each core (Figure 4.4).

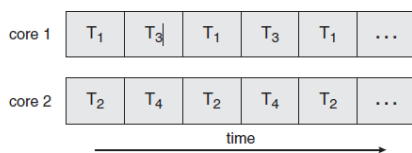


Figure 4.4 Parallel execution on a multicore system.



Figure 4.3 Concurrent execution on a single-core system.

Notice the distinction between *parallelism* and *concurrency* in this discussion. A system is parallel if it can perform more than one task simultaneously. In contrast, a concurrent system supports more than one task by allowing all the tasks to make progress. Thus, it is possible to have concurrency without parallelism. Before the advent of SMP and multicore architectures, most computer systems had only a single processor. CPU schedulers were designed to provide the illusion of parallelism by rapidly switching between processes in the system, thereby allowing each process to make progress. Such processes were running concurrently, but not in parallel.

As systems have grown from tens of threads to thousands of threads, CPU designers have improved system performance by adding hardware to improve thread performance. Modern Intel CPUs frequently support two threads per core, while the Oracle T4 CPU supports eight threads per core. This support means that multiple threads can be loaded into the core for fast switching. Multicore computers will no doubt continue to increase in core counts and hardware thread support.

1.8 Operating system overview-objectives and functions. ▲

Objectives and Functions

- A program that is executed by the processor that frequently relinquishes control and must depend on the processor to regain control.
 - A program that mediates between application programs and the hardware
 - A set of procedures that enable a group of people to use a computer system.
 - A program that controls the execution of application programs
 - An interface between applications and hardware

Functions

Usage

Computer system

Control

Support

Usage

- ❖ Users of a computer system:
 - ❖ Programs - use memory, use CPU time, use I/O devices
 - ❖ Human users
 - ❖ Programmers - use program development tools such as debuggers, editors and users - use application programs, e.g. Internet explorer

Computer system

hardware + software

OS is a part of the computer software, it is a program. It is a very special program, that is the first to be executed when the computer is switched on, and is supposed to control and support the execution of other programs and the overall usage of the computer system.

Control

The operating system controls the usage of the computer resources - hardware devices and software utilities. We can think of an operating system as a *Resource Manager*. Here are some of the resources managed by the OS:

- Processors,
- Main memory,
- Secondary Memory,
- Peripheral devices,
- Information.

Support

- ✓ The operating system provides a number of services to assist the users of the computer system:

For the programmers:

Utilities - debuggers, editors, file management, etc.

For the end users - provides the interface to the application programs

For programs - loads instructions and data into memory, prepares I/O devices for usage, handles interrupts and error conditions.

1.9 Evolution of Operating System

1.9. **Serial Processing** - 1940's – 1950's programmer interacted directly with hardware. No operating system.

Problems

Scheduling - users sign up for machine time. Wasted computing time

Setup Time- Setup included loading the compiler, source program, saving compiled program, and loading and linking. If an error occurred - start over.

1.9.2 Simple Batch Systems

Improve the utilization of computers.

Jobs were submitted on cards or tape to an operator who batches jobs together sequentially. The program that controls the execution of the jobs was called **monitor** - a simple version of an operating system. The interface to the monitor was accomplished through Job Control Language (JCL). For example, a JCL request could be to run the compiler for a particular programming language, then to link and load the program, then to run the user program.

Hardware features:

Memory protection: do not allow the memory area containing the monitor to be altered

Timer: prevents a job from monopolizing the system

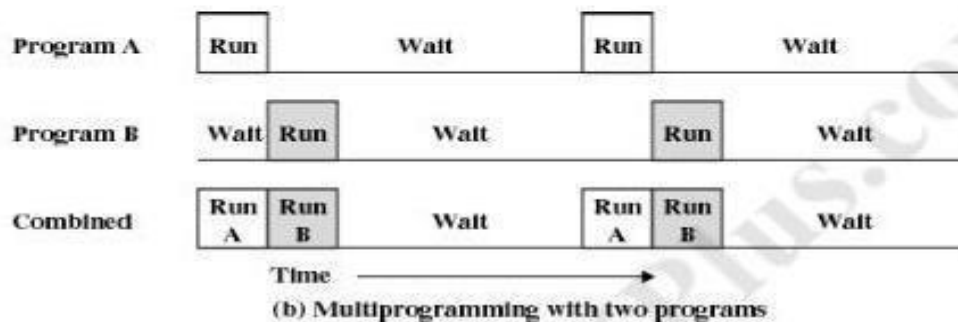
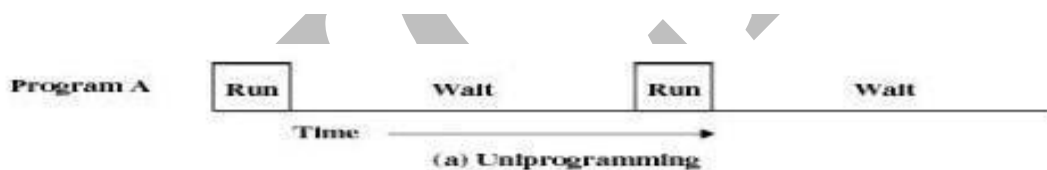
Problems:

Bad utilization of CPU time - the processor stays idle while I/O devices are in use.

1.9.3

Multiprogrammed Batch Systems

More than one program resides in the main memory. While a program A uses an I/O device the processor does not stay idle, instead it runs another program B.



New features:

Memory management - to have several jobs ready to run, they must be kept in main memory

Job scheduling - the processor must decide which program to run.

1.9.4 Time-Sharing Systems

Multiprogramming systems: Several programs use the computer system.

Time-sharing systems: Several (human) users use the computer system interactively.

Characteristics:

- Using multiprogramming to handle multiple interactive jobs
- Processor's time is shared among multiple users
- Multiple users simultaneously access the system through terminals

1.9.5 Operating-System Services

The OS provides certain services to programs and to the users of those programs.

1. Program execution:

The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).

2. I/O operations:

A running program may require I/O. This I/O may involve a file or an I/O device.

3. File-system manipulation:

The program needs to read, write, create and delete files.

4. Communications :

In many circumstances, one process needs to exchange information with another process. Such communication can occur in two major ways. The first takes place between processes that are executing on the same computer; the second takes place between processes that are executing on different computer systems that are tied together by a computer network.

5. Error detection:

The operating system constantly needs to be aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

6. Resource allocation:

Different types of resources are managed by the Os.

When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.

7. Accounting:

We want to keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting or simply for accumulating usage statistics.

8. Protection:

The owners of information stored in a multiuser computer system may want to control use of that information. Security of the system is also important.

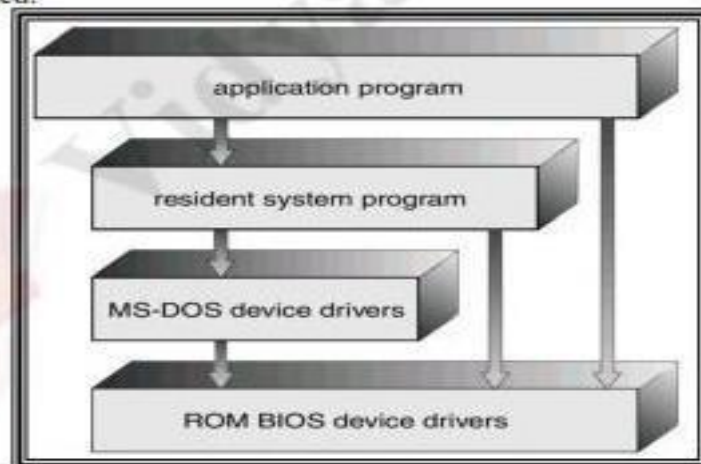
1.10 Computer System Organization

1.10.1 Operating System Structure and Operations

1.10. Operating System Structure

1.10.1.1 MS-DOS System Structure

- ✓ MS-DOS – written to provide the most functionality in the least space.
- ✓ Not divided into modules.
- ✓ Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated.



1.10.1.2 Unix System Structure

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts.

- **Systems programs** – use kernel supported system calls to provide useful functions such as compilation and file manipulation.
- **The kernel** - Consists of everything below the system-call interface and above the physical hardware

1.10.1.3 Layered Approach

- ✓ The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- ✓ An OS layer is an implementation of an abstract object that is the encapsulation of data and operations that can manipulate those data. These operations (routines) can be invoked by higher-level layers. The layer itself can invoke operations on lower-level layers.
- ✓ Layered approach provides modularity. With modularity, layers are selected such that each layer uses functions (operations) and services of only lower-level layers.
- ✓ Each layer is implemented by using only those operations that are provided lower level layers.
- ✓ The major difficulty is appropriate definition of various layers.

1.10.1.4 Microkernel System Structure

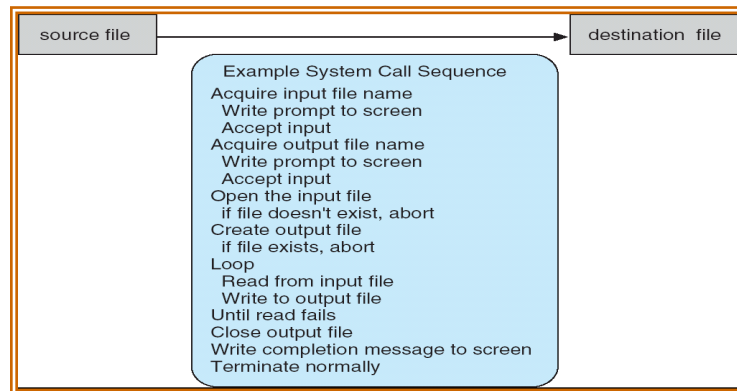
- ✓ Moves as much from the kernel into “user” space.
- ✓ Communication takes place between user modules using message passing.
 - ❖ Benefits:
 - Easier to extend a microkernel
 - Easier to port the operating system to new architectures

1.10. Operating-System Operations

- ✓ If there are no processes to execute, no I/O devices to service, and no users to whom to respond, an operating system will sit quietly, waiting for something to happen. Events are almost always signaled by the occurrence of an interrupt or a trap.
- ✓ A trap (or an exception) is a software-generated interrupt caused either by an error (for example, division by zero or invalid memory access) or by a specific request from a user program that an operating-system service be performed. The interrupt-driven nature of an operating system defines that system’s general structure.
- ✓ Without protection against these sorts of errors, either the computer must execute only one process at a time or all output must be suspect. A properly designed operating system must ensure that an incorrect (or malicious) program cannot cause other programs to execute incorrectly.

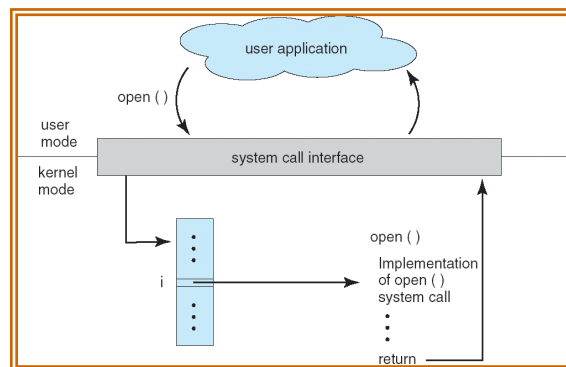
✓ **System Calls**

- **Programming interface to the services provided by the OS**
- **Typically written in a high-level language (C or C++)**
- **Mostly accessed by programs via a high-level Application Program Interface (API) rather than direct system call use**
- **Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)**
- **System call sequence to copy the contents of one file to another file**



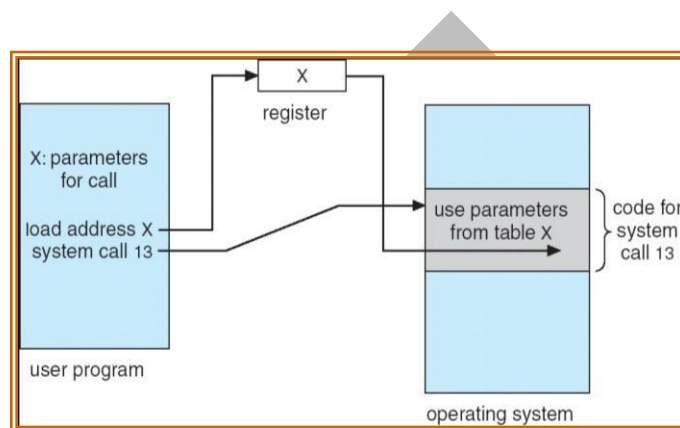
1.11.1 System Call Implementation

- **Typically, a number associated with each system call**
- ✓ **System-call interface maintains a table indexed according to these numbers**
- **The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values**
- **The caller need know nothing about how the system call is implemented**
- ✓ **Just needs to obey API and understand what OS will do as a result call**
- ✓ **Most details of OS interface hidden from programmer by API**
- ✓ **Managed by run-time support library (set of functions built into libraries included with compiler)**



1.11.2 System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
 - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
 - Simplest: pass the parameters in *registers*
 - In some cases, may be more parameters than registers
 - Parameters stored in a *block*, or table, in memory, and address of block passed as a parameter in a register
 - This approach taken by Linux and Solaris
 - Parameters placed, or *pushed*, onto the *stack* by the program and *popped* off the stack by the operating system
 - Block and stack methods do not limit the number or length of parameters being passed



Types of System Calls

- **Process control**
- **File management**
- **Device management**
- **Information maintenance**
- **Communications**

1.12 System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
 - File manipulation
 - Status information
 - File modification

- Programming language support
 - Program loading and execution
 - Communications
 - Application programs
- Most users' view of the operation system is defined by system programs, not the actual systemcalls
 - Provide a convenient environment for program development and execution
 - Some of them are simply user interfaces to system calls; others are considerably more complex
 - File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
 - Status information
 - Some ask the system for info - date, time, amount of available memory, disk space, number of users
 - Others provide detailed performance, logging, and debugging information
 - Typically, these programs format and print the output to the terminal or other output devices
 - Some systems implement a registry - used to store and retrieve configuration information
 - File modification
 - Text editors to create and modify files
 - Special commands to search contents of files or perform transformations of the text
 - Programming-language support - Compilers, assemblers, debuggers and interpreters sometimes provided
 - Program loading and execution- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
 - Communications - Provide the mechanism for creating virtual connections among processes, users, and computer systems
 - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

1.12 OS Generation and System Boot.

1.12.1 OS Generation

Historically operating systems have been tightly related to the computer architecture, it is good idea to study the history of operating systems from the architecture of the computers on which they run.

Operating systems have evolved through a number of distinct phases or generations which corresponds roughly to the decades.

The 1940's - First Generations

The earliest electronic digital computers had no operating systems. Machines of the time were so primitive that programs were often entered one bit at time on rows of mechanical switches (plug boards). Programming languages were unknown (not even assembly languages). Operating systems were unheard of .

The 1950's - Second Generation

By the early 1950's, the routine had improved somewhat with the introduction of punch cards. The General Motors Research Laboratories implemented the first operating systems in early 1950's for their IBM 701. The system of the 50's generally ran one job at a time. These were called single-stream batch processing systems because programs and data were submitted in groups or batches.

The 1960's - Third Generation

The systems of the 1960's were also batch processing systems, but they were able to take better advantage of the computer's resources by running several jobs at once. So operating systems designers developed the concept of multiprogramming in which several jobs are in main memory at once; a processor is switched from job to job as needed to keep several jobs advancing while keeping the peripheral devices in use.

For example, on the system with no multiprogramming, when the current job paused to wait for other I/O operation to complete, the CPU simply sat idle until the I/O finished. The solution for this problem that evolved was to partition memory into several pieces, with a different job in each partition. While one job was waiting for I/O to complete, another job could be using the CPU.

Another major feature in third-generation operating system was the technique called spooling (simultaneous peripheral operations on line). In spooling, a high-speed device like a disk interposed between a running program and a low-speed device involved with the program in input/output. Instead of writing directly to a printer, for example, outputs are written to the disk. Programs can run to completion faster, and other programs can be initiated sooner when the printer becomes available, the outputs may be printed.

Note that spooling technique is much like thread being spun to a spool so that it may be later be unwound as needed.

Another feature present in this generation was time-sharing technique, a variant of multiprogramming technique, in which each user has an on-line (i.e., directly connected) terminal. Because the user is

present and interacting with the computer, the computer system must respond quickly to user requests, otherwise user productivity could suffer. Timesharing systems were developed to multiprogram large number of simultaneous interactive users.

Fourth Generation

With the development of LSI (Large Scale Integration) circuits, chips, operating system entered in the system entered in the personal computer and the workstation age. Microprocessor technology evolved to the point that it become possible to build desktop computers as powerful as the mainframes of the 1970s. Two operating systems have dominated the personal computer scene: MS-DOS, written by Microsoft, Inc. for the IBM PC and other machines using the Intel 8088 CPU and its successors, and UNIX, which is dominant on the large personal computers using the Motorola 6899 CPU family.

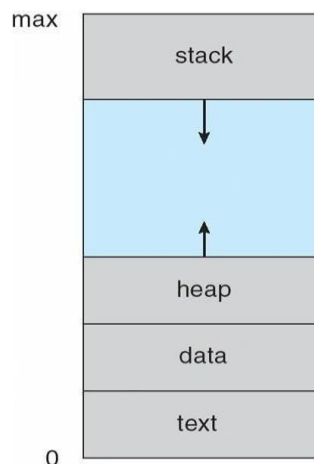
12.2 System Boot.

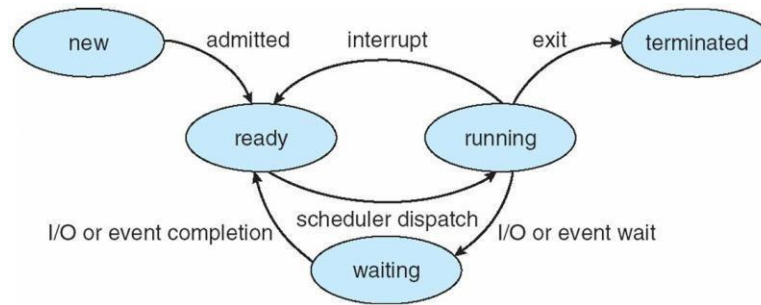
- Operating system must be made available to hardware so hardware can start it
 - Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it
 - Sometimes two-step process where **boot block** at fixed location loads bootstrap loader
 - When power initialized on system, execution starts at a fixed memory location
 - Firmware used to hold initial boot code

CS8493- Operating System
UNIT-II
PROCESS MANAGEMENT

Processes-Process Concept:

- An operating system executes a variety of programs:
 - Batch system – —**jobs**”
 - Time-shared systems – —**user programs**” or —**tasks**”
- We will use the terms *job* and *process* almost interchangeably
- **Process** – is a program in execution (informal definition)
- Program is *passive* entity stored on disk (**executable file**), process is *active*
 - Program becomes process when executable file loaded into memory
- Execution of program started via GUI, command line entry of its name, etc
- One program can be several processes
 - Consider multiple users executing the same program
- In memory, a process consists of **multiple parts:**
 - **Program code**, also called **text section**
 - **Current activity** including
 - **program counter**
 - processor registers
 - **Stack** containing temporary data
 - Function parameters return addresses, local variables
 - **Data section** containing global variables
 - **Heap** containing memory dynamically allocated during run time





- As a process executes, it changes **state**
 - **new**: The process is being created
 - **ready**: The process is waiting to be assigned to a processor
 - **running**: Instructions are being executed
 - **waiting**: The process is waiting for some event to occur
 - **terminated**: The process has finished execution

PROCESS CONTROL BLOCK (PCB)

Each process is represented in the operating system by a **process control block (PCB)**—also called a **task control block**. A PCB is shown in 3.3. It contains many pieces of information associated with a specific process, including these:



Figure 3.3 Process control block (PCB)

- **Process state.** The state may be new, ready, running, waiting, halted, and so on.
- **Program counter:** The counter indicates the address of the next instruction to be executed for this process.
- **CPU registers:** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward (Figure 3.4).
- **CPU-scheduling information:** This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

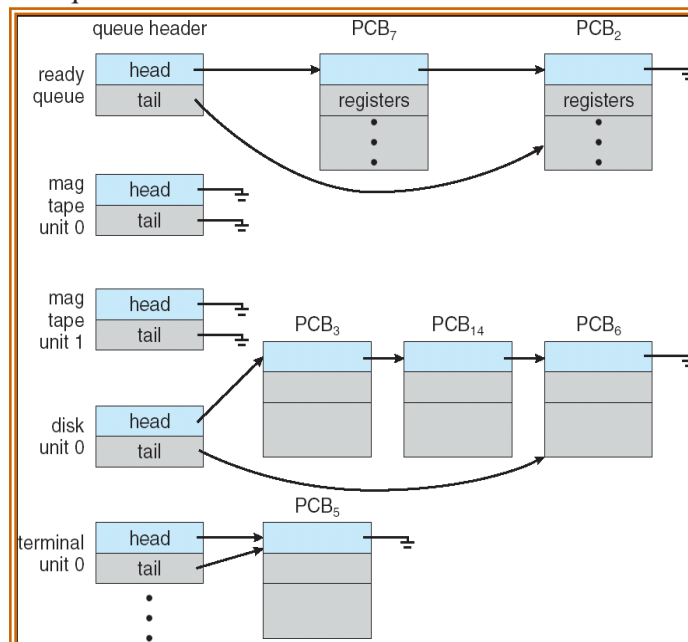
- **Memory-management information:** This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system
- **Accounting information.** This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- **I/O status information.** This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

Process Scheduling:

The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. the **process scheduler** selects an available process (possibly from a set of several available processes) for program execution on the CPU. For a single-processor system, there will never be more than one running process. If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.

Scheduling Queues

- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
- **Device queues** – set of processes waiting for an I/O device Processes migrate among the various queues.

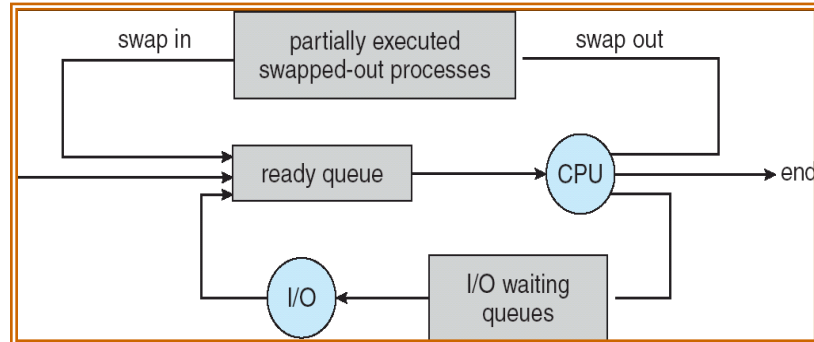


A common representation of process scheduling is a **queueing diagram**. Two types of queues are present: the ready queue and a set of device queues. The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system. A new process is initially put in the ready queue. It waits there until it is selected for execution, or **dispatched**. Once the process is allocated the CPU and is executing, one of several events could occur:

- The process could issue an I/O request and then be placed in an I/O queue.
- The process could create a new child process and wait for the child's termination.
- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

Schedulers

- **Long-term scheduler (or job scheduler)** – selects which processes should be brought into the ready queue
- **Short-term scheduler (or CPU scheduler)** – selects which process should be executed next and allocates CPU



- Short-term scheduler is invoked very frequently (milliseconds) → (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) → (may be slow)
- The long-term scheduler controls the *degree of multiprogramming*
- Processes can be described as either:
 - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
 - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

Some operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling. The key idea behind a medium-term scheduler is that sometimes it can be advantageous to remove a process from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming.

Context Switch

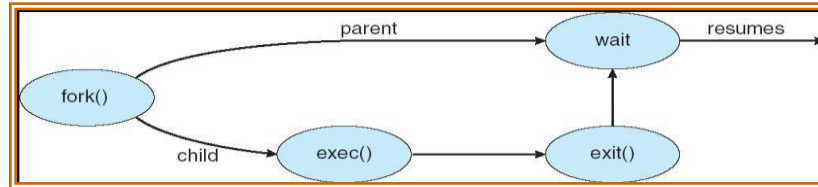
- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

Operations on Processes

Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
 - Parent and children share all resources
 - Children share subset of parent's resources
 - Parent and child share no resources
- Execution
 - Parent and children execute concurrently
 - Parent waits until children terminate
- Address space

- Child duplicate of parent
- Child has a program loaded into it
- UNIX examples
 - **fork** system call creates new process
 - **exec** system call used after a **fork** to replace the process' memory space with a new program



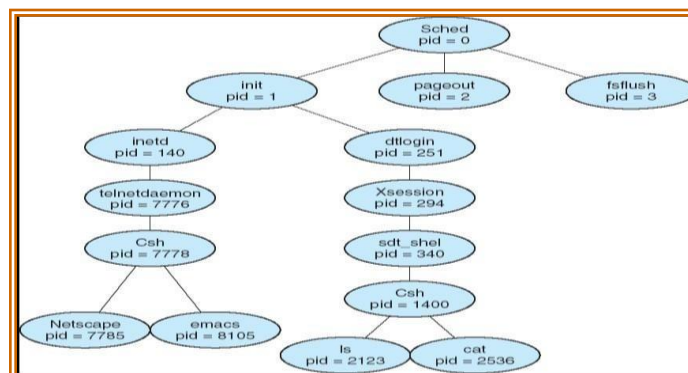
C Program Forking Separate Process

```

int main()
{
pid_t pid;
  /* fork another process */
  pid = fork();
  if (pid < 0) { /* error occurred */
    fprintf(stderr, "Fork Failed");
    exit(-1);
  }
  else if (pid == 0) { /* child process */
    execlp("/bin/ls", "ls", NULL);
  }
  else { /* parent process */
    /* parent will wait for the child to complete */
    wait (NULL);
    printf ("Child Complete");
    exit(0);
  }
}

```

A tree of processes on a typical Solaris



Process Termination

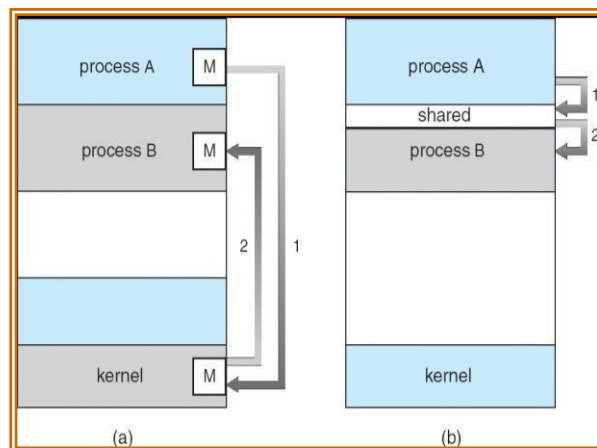
- Process executes last statement and asks the operating system to delete it (**exit**)
 - Output data from child to parent (via **wait**)
 - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
 - Child has exceeded allocated resources
 - Task assigned to child is no longer required
 - If parent is exiting
 - Some operating system do not allow child to continue if its parent terminates
 - All children terminated - *cascading termination*

Cooperating Processes

- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
 - Information sharing
 - Computation speed-up
 - Modularity
 - Convenience

Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
 - **send**(*message*) – message size fixed or variable
 - **receive**(*message*)
- If P and Q wish to communicate, they need to:
 - establish a *communication link* between them
 - exchange messages via send/receive
- Implementation of communication link
 - physical (e.g., shared memory, hardware bus)
 - logical (e.g., logical properties)



Direct Communication

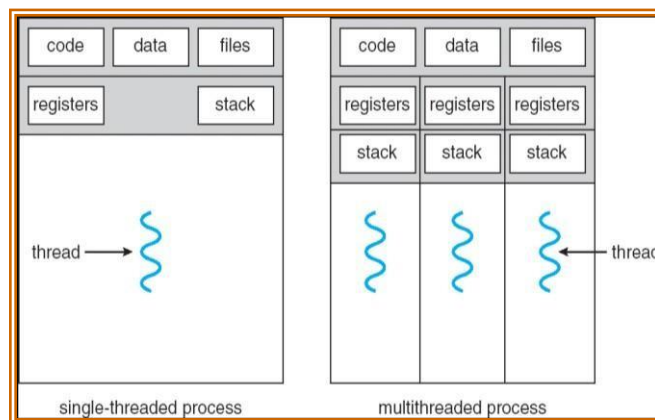
- Processes must name each other explicitly:
 - **send** ($P, message$) – send a message to process P
 - **receive**($Q, message$) – receive a message from process Q
- Properties of communication link
 - Links are established automatically
 - A link is associated with exactly one pair of communicating processes
 - Between each pair there exists exactly one link
 - The link may be unidirectional, but is usually bi-directional

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
 - Each mailbox has a unique id
 - Processes can communicate only if they share a mailbox
- Properties of communication link
 - Link established only if processes share a common mailbox
 - A link may be associated with many processes
 - Each pair of processes may share several communication links
 - Link may be unidirectional or bi-directional

Threads- Overview

A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals. A traditional (or *heavyweight*) process has a single thread of control. If a process has multiple threads of control, it can perform more than one task at a time.



Benefits

The benefits of multithreaded programming can be broken down into four major categories:

1. **Responsiveness.** Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to the user.

2. **Resource sharing.** Processes can only share resources through techniques such as shared memory and message passing.
3. **Economy.** Allocating memory and resources for process creation is costly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads.
4. **Scalability.** The benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores.

Multicore Programming

Earlier in the history of computer design, in response to the need for more computing performance, single-CPU systems evolved into multi-CPU systems. A more recent, similar trend in system design is to place multiple computing cores on a single chip. Each core appears as a separate processor to the operating system. Whether the cores appear across CPU chips or within CPU chips, we call these systems **multicore** or **multiprocessor** systems.

Multithreaded programming provides a mechanism for more efficient use of these multiple computing cores and improved concurrency. Consider an application with four threads. On a system with a single computing core, concurrency merely means that the execution of the threads will be interleaved over time because the processing core is capable of executing only one thread at a time. On a system with multiple cores, however,

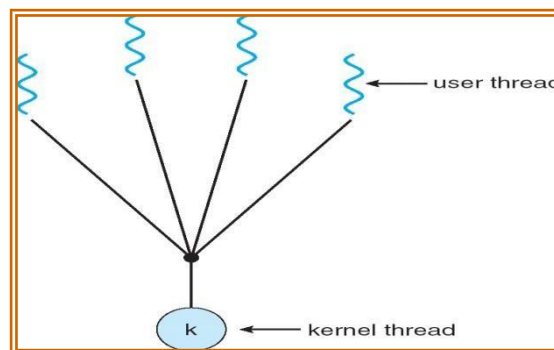
Concurrency means that the threads can run in parallel, because the system can assign a separate thread to each core. Notice the distinction between *parallelism* and *concurrency* in this discussion. A system is parallel if it can perform more than one task simultaneously. In contrast, a concurrent system supports more than one task by allowing all the tasks to make progress.

Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many

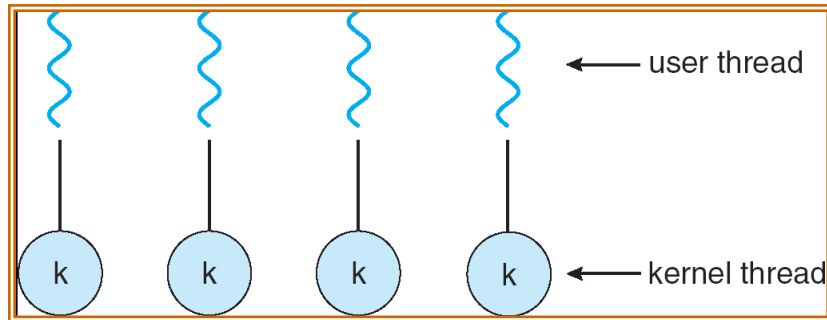
1. Many-to-One

- Many user-level threads mapped to single kernel thread
- Examples:
 - Solaris Green Threads
 - GNU Portable Threads



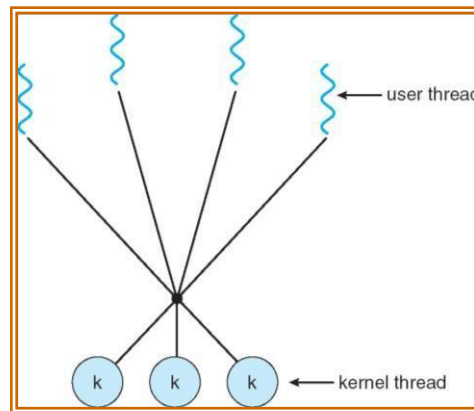
2. One-to-One

- Each user-level thread maps to kernel thread
- Examples
 - Windows NT/XP/2000
 - Linux
 - Solaris 9 and later



3. Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows NT/2000 with the *ThreadFiber* package



Windows 7

Windows implements the Windows API, which is the primary API for the family of Microsoft operating systems (Windows 98, NT, 2000, and XP, as well as Windows 7). Indeed, much of what is mentioned in this section applies to this entire family of operating systems. A Windows application runs as a separate process, and each process may contain one or more threads.

The general components of a thread include:

- A thread ID uniquely identifying the thread
- A register set representing the status of the processor
- A user stack, employed when the thread is running in user mode, and a kernel stack, employed when the thread is running in kernel mode

- A private storage area used by various run-time libraries and dynamic link libraries (DLLs).

The register set, stacks, and private storage area are known as the **context** of the thread. The primary data structures of a thread include:

- ETHREAD—executive thread block
- KTHREAD—kernel thread block
- TEB—thread environment block

Process Synchronization

- Concurrent access to shared data may result in data inconsistency.
- Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.
- Shared-memory solution to bounded-buffer problem allows at most $n - 1$ items in buffer at the same time. A solution, where all N buffers are used is not simple.
- Suppose that we modify the producer-consumer code by adding a variable *counter*, initialized to 0 and increment it each time a new item is added to the buffer
- **Race condition:** The situation where several processes access – and manipulate shared data concurrently. The final value of the shared data depends upon which process finishes last.
- To prevent race conditions, **concurrent processes** must be **synchronized**.

The Critical-Section Problem:

- There are n processes that are competing to use some shared data
- Each process has a code segment, called critical section, in which the shared data is accessed.
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.

Requirements to be satisfied for a Solution to the Critical-Section Problem:

1. **Mutual Exclusion** - If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.
2. **Progress** - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
3. **Bounded Waiting** - A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

```
do {
    entry section
    critical section
    exit section
    remainder section
} while (true);
```

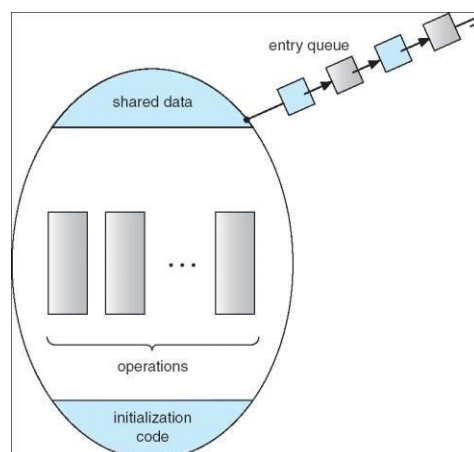
- Two general approaches are used to handle critical sections in operating systems: **preemptive kernels** and **nonpreemptive kernels**.
- A preemptive kernel allows a process to be preempted while it is running in kernel mode.
- A non-preemptive kernel does not allow a process running in kernel mode to be preempted; a kernel-mode process will run until it exits kernel mode, blocks, or voluntarily yields control of the CPU.

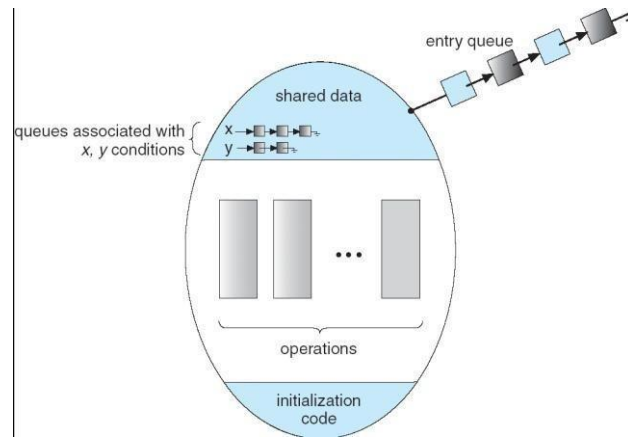
Mutex Locks

- A high-level abstraction that provides a convenient and effective mechanism for process synchronization
- Only one process may be active within the monitor at a time monitor monitor-name

```
{  
  
// shared variable declarations  
  
procedure body P1 (...) { .... }  
  
...  
  
procedure body Pn (...) {.....}  
  
{  
  
initialization code  
  
}  
  
}
```

- To allow a process to wait within the monitor, a condition variable must be declared as a condition x, y;
- Two operations on a condition variable:
 - x.wait ()—a process that invokes the operation is suspended.
 - x.signal ()—resumes one of the suspended processes(if any)





Solution to Dining Philosophers Problem

Monitor DP

```

{
enum { THINKING; HUNGRY, EATING) state [5] ;
condition self [5];
void pickup (int i) {
state[i] = HUNGRY;
test(i);
if (state[i] != EATING) self [i].wait;
}
void putdown (int i) {
state[i] = THINKING;
// test left and right neighbors
test((i + 4) % 5);
test((i + 1) % 5);
}
void test (int i) {
if ( (state[(i + 4) % 5] != EATING) &&
(state[i] == HUNGRY) &&
(state[(i + 1) % 5] != EATING) ) {
state[i] = EATING ;
self[i].signal () ;
}
}
initialization_code() {
for (int i = 0; i < 5; i++)
state[i] = THINKING;
}
}

```

Semaphores

- It is a synchronization tool that is used to generalize the solution to the critical section problem in complex situations.
- A Semaphore s is an integer variable that can only be accessed via two indivisible (atomic) operations namely

```
wait (s)
{
1. wait or P operation ( to test )
2. signal or V operation ( to increment )
while(s <= 0);
s--;
}
signal (s)
{
s++;
}
```

Mutual Exclusion Implementation using semaphore

```
do
{
wait(mutex);
critical section

remainder section
} while (1);
signal(mutex);
```

Semaphore Implementation

- The semaphore discussed so far requires a busy waiting. That is if a process is in critical-section, the other process that tries to enter its critical-section must loop continuously in the entry code.
- To overcome the busy waiting problem, the definition of the semaphore operations wait and signal should be modified.
 - When a process executes the wait operation and finds that the semaphore value is not positive, the process can block itself. The block operation places the process into a waiting queue associated with the semaphore.
 - A process that is blocked waiting on a semaphore should be restarted when some other process executes a signal operation. The blocked process should be restarted by a wakeup operation which put that process into ready queue.
- To implemented the semaphore, we define a semaphore as a record as:

```
typedef struct {
int value;
struct process *L;
} semaphore;
```

Deadlock & starvation:

Example: Consider a system of two processes , P0 & P1 each accessing two semaphores ,S & Q, set to the value 1.

P0	P1
Wait (S)	Wait (Q)
Wait (Q)	Wait (S)
.	.
.	.
.	.
Signal(S)	Signal(Q)
Signal(Q)	Signal(S)

- Suppose that P0 executes wait(S), then P1 executes wait(Q). When P0 executes wait(Q), it must wait until P1 executes signal(Q). Similarly when P1 executes wait(S), it must wait until P0 executes signal(S). Since these signal operations cannot be executed, P0 & P1 are deadlocked.
- Another problem related to deadlock is indefinite blocking or starvation, a situation where a process wait indefinitely within the semaphore. Indefinite blocking may occur if we add or remove processes from the list associated with a semaphore in LIFO order.

Types of Semaphores

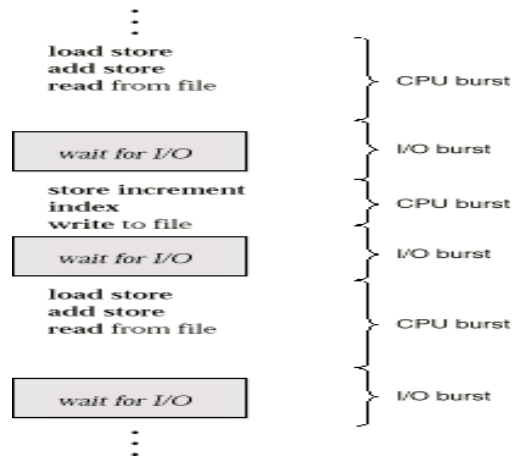
- *Counting* semaphore – any positive integer value
- *Binary* semaphore – integer value can range only between 0 and 1

CPU Scheduling

- CPU scheduling is the basis of multi programmed operating systems.
- The objective of multiprogramming is to have some process running at all times, in order to maximize CPU utilization.
- Scheduling is a fundamental operating-system function.
- Almost all computer resources are scheduled before use.

CPU-I/O Burst Cycle

- Process execution consists of a **cycle** of CPU execution and I/O wait.
- Processes alternate between these two states.
- Process execution begins with a **CPU burst**.
- That is followed by an **I/O burst**, then another CPU burst, then another I/O burst, and so on.
- Eventually, the last CPU burst will end with a system request to terminate execution, rather than with another I/O burst.



CPU Scheduler

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed.

The selection process is carried out by the **short-term scheduler** (or CPU scheduler).

The ready queue is not necessarily a first-in, first-out (FIFO) queue. It may be a FIFO queue, a priority queue, a tree, or simply an unordered linked list.

Preemptive Scheduling

- CPU scheduling decisions may take place under the following four circumstances:
 1. When a process switches from the running state to the waiting state
 2. When a process switches from the running state to the ready state
 3. When a process switches from the waiting state to the ready state
 4. When a process terminates
- Under 1 & 4 scheduling scheme is non preemptive.
- Otherwise the scheduling scheme is preemptive.

Non-preemptive Scheduling

- In non preemptive scheduling, once the CPU has been allocated a process, the process keeps the CPU until it releases the CPU either by termination or by switching to the waiting state.
- This scheduling method is used by the Microsoft windows environment.

Dispatcher

The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler.

This function involves:

1. Switching context
2. Switching to user mode
3. Jumping to the proper location in the user program to restart that program

Scheduling Criteria

1. CPU utilization: The CPU should be kept as busy as possible. CPU utilization may range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily used system).

2. Throughput: It is the number of processes completed per time unit. For long processes, this rate may be 1 process per hour; for short transactions, throughput might be 10 processes per second.

3. Turnaround time: The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.

4. Waiting time: Waiting time is the sum of the periods spent waiting in the ready queue.

5. Response time: It is the amount of time it takes to start responding, but not the time that it takes to output that response.

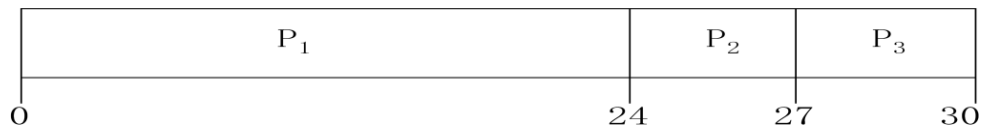
CPU Scheduling Algorithms

1. First-Come, First-Served Scheduling
2. Shortest Job First Scheduling
3. Priority Scheduling
4. Round Robin Scheduling

First-Come, First-Served (FCFS) Scheduling

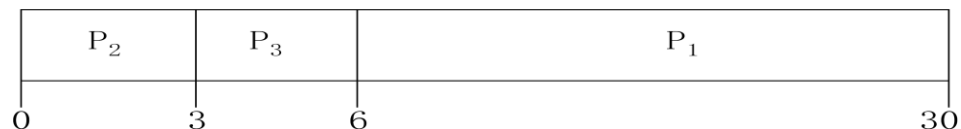
<u>Process</u>	<u>Burst Time</u>
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1, P_2, P_3
The Gantt Chart for the schedule is:



- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

Suppose that the processes arrive in the order



The Gantt chart for the schedule is:

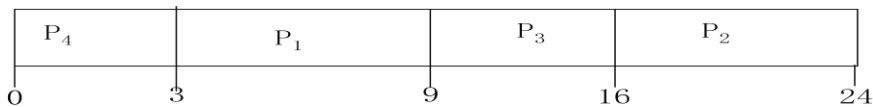
- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- *Convoy effect* short process behind long process

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
 - The difficulty is knowing the length of the next CPU request

Process	Arrival Time	Burst Time
P_1	0.0	6
P_2	2.0	8
P_3	4.0	7
P_4	5.0	3

- SJF scheduling chart

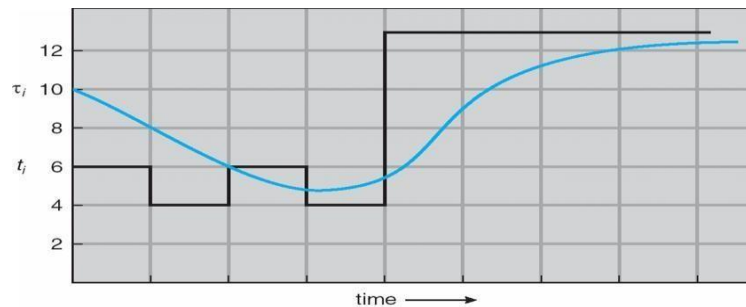


- Average waiting time = $(3 + 16 + 9 + 0) / 4 = 7$

Determining Length of Next CPU Burst

- Can only estimate the length
- Can be done by using the length of previous CPU bursts, using exponential averaging

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 < \alpha < 1$
4. Define: $\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n$



CPU burst (t_i)	6	4	6	4	13	13	13	...	
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

Examples of Exponential Averaging

- $\alpha = 0$
 - $\tau_{n+1} = \tau_n$
 - Recent history does not count
- $\alpha = 1$
 - $\tau_{n+1} = \alpha \tau_n$
 - Only the actual last CPU burst counts
- If we expand the formula, we get:

$$\begin{aligned}\tau_{n+1} &= \alpha \tau_n + (1 - \alpha) \alpha \tau_{n-1} + \dots \\ &+ (1 - \alpha)^j \alpha \tau_{n-j} + \dots \\ &+ (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ÷ highest priority)
 - Preemptive
 - nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem ÷ **Starvation** – low priority processes may never execute
- Solution ÷ **Aging** – as time progresses increase the priority of the process

Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q , then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Performance
 - q large \rightarrow FIFO

- q small $\rightarrow q$ must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

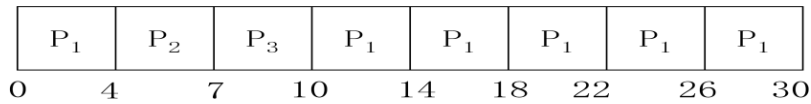
Process Burst Time

P_1 24

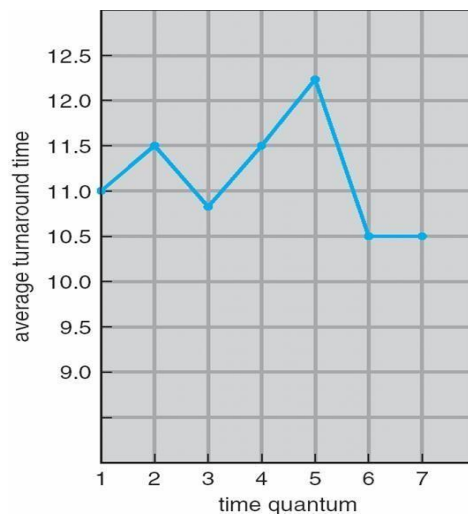
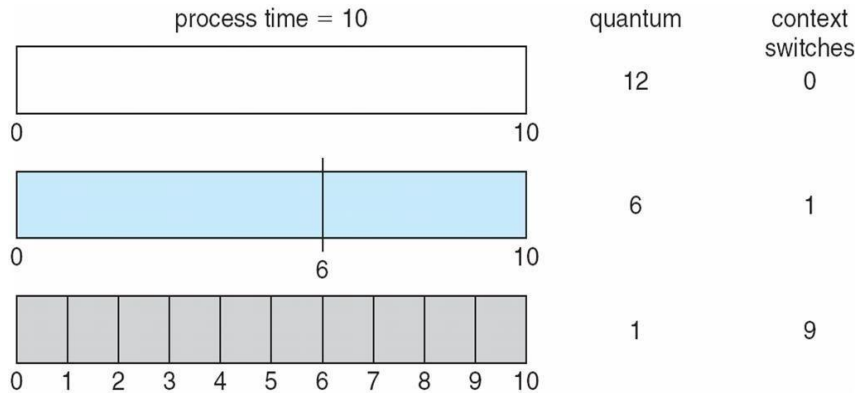
P_2 3

P_3 3

- The Gantt chart is:



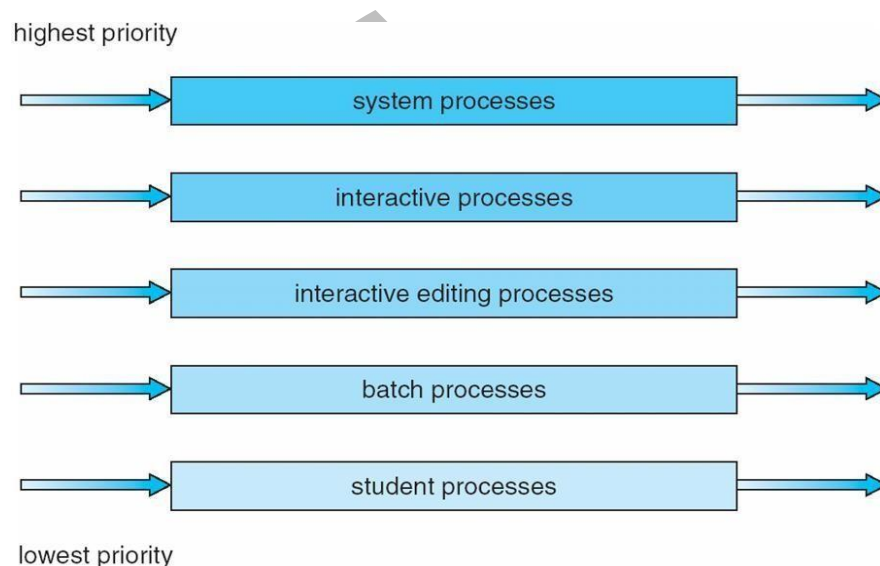
- Typically, higher average turnaround than SJF, but better *response*



process	time
P_1	6
P_2	3
P_3	1
P_4	7

Multilevel Queue

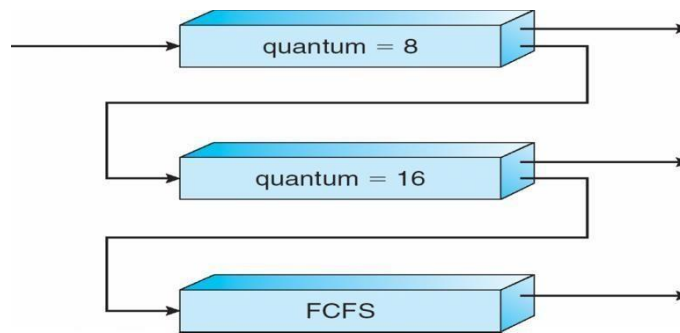
- Ready queue is partitioned into separate queues:
foreground (interactive)
background (batch)
- Each queue has its own scheduling algorithm
 - foreground – RR
 - background – FCFS
- Scheduling must be done between the queues
 - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
 - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
 - 20% to background in FCFS



Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
 - number of queues
 - scheduling algorithms for each queue
 - method used to determine when to upgrade a process
 - method used to determine when to demote a process

- method used to determine which queue a process will enter when that process needs service



Deadlocks

- A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.
- Example
 - System has 2 disk drives.
 - P_1 and P_2 each hold one disk drive and each needs another one.
- Example
 - semaphores A and B , initialized to 1

P_0	P_1
wait (A);	wait(B)
wait (B);	wait(A)

System Model

- Resource types R_1, R_2, \dots, R_m
CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
 - request
 - use
 - release

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.




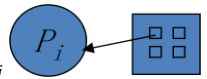
- **Mutual exclusion:** only one process at a time can use a resource.
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.

- **Circular wait:** there exists a set $\{P_0, P_1, \dots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1 , P_1 is waiting for a resource that is held by P_2, \dots, P_{n-1} is waiting for a resource that is held by P_n , and P_0 is waiting for a resource that is held by P_0 .

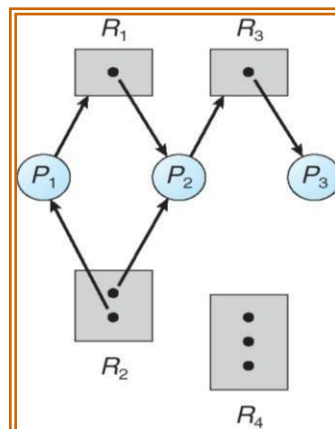
Resource-Allocation Graph

A set of vertices V and a set of edges E .

- V is partitioned into two types:
 - $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the processes in the system.
 - $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_i \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$

- Process 
- Resource Type with 4 instances 
- P_i requests instance of R_j 
- P_i is holding an instance of R_j 

Example of a Resource Allocation Graph



Basic Facts

- If graph contains no cycles \rightarrow no deadlock.
- If graph contains a cycle \rightarrow
 - if only one instance per resource type, then deadlock.

- if several instances per resource type, possibility of deadlock.

Deadlock Prevention

- Mutual Exclusion – not required for sharable resources; must hold for non-sharable resources.
- Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources.
 - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none.
 - Low resource utilization; starvation possible.
- **No Preemption** –
 - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.
 - Preempted resources are added to the list of resources for which the process is waiting.
 - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.
- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

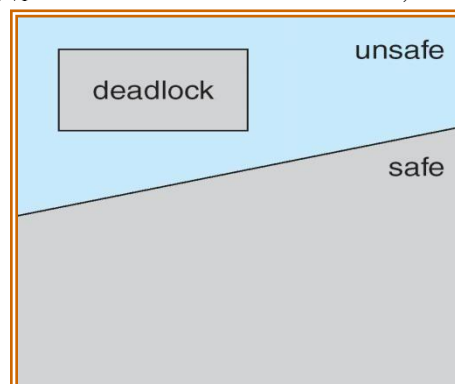
Deadlock Avoidance

Requires that the system has some additional *a priori* information available.

- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need.
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes.

Safe State

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.
- System is in safe state if there exists a sequence $\langle P_1, P_2, \dots, P_n \rangle$ of ALL the processes in the systems such that for each P_i , the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j , with $j < i$.
- That is:
 - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
 - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
 - When P_i terminates, P_{i+1} can obtain its needed resources, and soon.

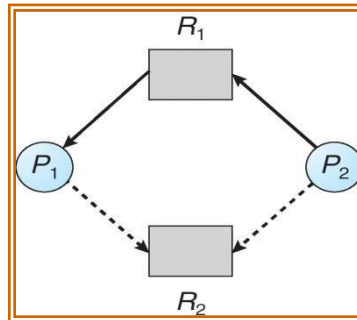


Avoidance algorithms

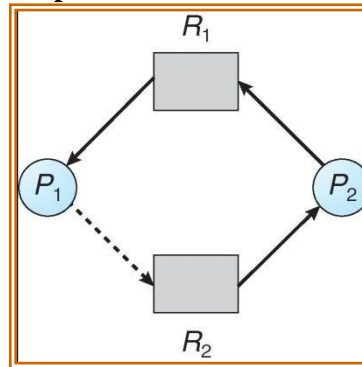
- Single instance of a resource type. Use a resource-allocation graph
- Multiple instances of a resource type. Use the banker's algorithm

Resource-Allocation Graph Scheme

- *Claim edge* $P_i \rightarrow R_j$ indicated that process P_j may request resource R_j ; represented by a dashed line.
- Claim edge converts to request edge when a process requests a resource.
- Request edge converted to an assignment edge when the resource is allocated to the process.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- Resources must be claimed *a priori* in the system.



Unsafe State In Resource-Allocation Graph



Banker's Algorithm

- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.
- Let n = number of processes, and m = number of resources types.
 - **Available:** Vector of length m . If available $[j] = k$, there are k instances of resource type R_j available.
 - **Max:** $n \times m$ matrix. If $Max [i,j] = k$, then process P_i may request at most k instances of resource type R_j .
 - **Allocation:** $n \times m$ matrix. If $Allocation[i,j] = k$ then P_i is currently allocated k instances of R_j .
 - **Need:** $n \times m$ matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task.
 $Need [i,j] = Max[i,j] - Allocation [i,j]$.

Example of Banker's Algorithm

- 5 processes P_0 through P_4 ;

3 resource types:

A (10 instances), B (5 instances), and C (7 instances).

- Snapshot at time T_0 :

	<i>Allocation</i>			<i>Max</i>			<i>Available</i>		
	<i>A</i>	<i>B</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>C</i>
P_0	0	1	0	7	5	3	3	3	2
P_1	2	0	0	3	2	2			
P_2	3	0	2	9	0	2			
P_3	2	1	1	2	2	2			
P_4	0	0	2	4	3	3			

- The content of the matrix *Need* is defined to be $Max - Allocation$.

	<i>Need</i>		
	<i>A</i>	<i>B</i>	<i>C</i>
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

- The system is in a safe state since the sequence $\langle P_1, P_3, P_4, P_2, P_0 \rangle$ satisfies safety criteria.

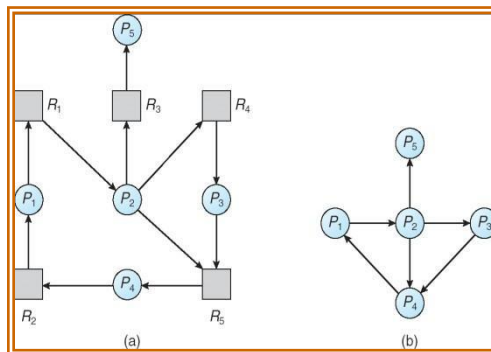
Deadlock Detection

- **Allow system to enter deadlock state**
- **Detection algorithm**
- **Recovery scheme**

Single Instance of Each Resource Type

- Maintain *wait-for* graph

- Nodes are processes.
 - $P_i \rightarrow P_j$ if P_i is waiting for P_j .
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock.
 - An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph.



Several Instances of a Resource Type

- *Available*: A vector of length m indicates the number of available resources of each type.
- *Allocation*: An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.
- *Request*: An $n \times m$ matrix indicates the current request of each process. If $Request[i_j] = k$, then process P_i is requesting k more instances of resource type R_j .

Detection Algorithm

1. Let *Work* and *Finish* be vectors of length m and n , respectively Initialize:

(a) $Work = Available$

(b) For $i = 1, 2, \dots, n$, if $Allocation_i \neq 0$, then $Finish[i] = false$; otherwise, $Finish[i] = true$.

2. Find an index i such that both:

(a) $Finish[i] == false$

(b) $Request_i \leq Work$

If no such i exists, go to step 4.

3. $Work = Work + Allocation_i$

$Finish[i] = true$

go to step 2.

4. If $Finish[i] == false$, for some $i, 1 \leq i \leq n$, then the system is in deadlock state. Moreover, if $Finish[i] == false$, then P_i is deadlocked.

Example of Detection Algorithm

- Five processes P_0 through P_4 ; three resource types A (7 instances), B (2 instances), and C (6 instances).
- Snapshot at time T_0 :

	<i>Allocation</i>			<i>Request Available</i>		
	<i>A</i>	<i>B</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>C</i>
P_0	0	1	0	0	0	0
P_1	2	0	0	2	0	2
P_2	3	0	3	0	0	0
P_3	2	1	1	1	0	0
P_4	0	0	2	0	0	2

- Sequence $\langle P_0, P_2, P_3, P_1, P_4 \rangle$ will result in $Finish[i] = true$ for all i .
- P_2 requests an additional instance of type C.

	<i>Request</i>		
	<i>A</i>	<i>B</i>	<i>C</i>
P_0	0	0	0
P_1	2	0	1
P_2	0	0	1
P_3	1	0	0
P_4	0	0	2

- State of system?
 - Can reclaim resources held by process P_0 , but insufficient resources to fulfill other processes; requests.
 - Deadlock exists, consisting of processes P_1, P_2, P_3 , and P_4 .

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated.

- In which order should we choose to abort?
 - Priority of the process.
 - How long process has computed, and how much longer to completion.
 - Resources the process has used.
 - Resources process needs to complete.
 - How many processes will need to be terminated.
 - Is process interactive or batch?
- Selecting a victim – minimize cost.
- Rollback – return to some safe state, restart process for that state.
- Starvation – same process may always be picked as victim, include number of rollback in cost factor.

UNIT-III

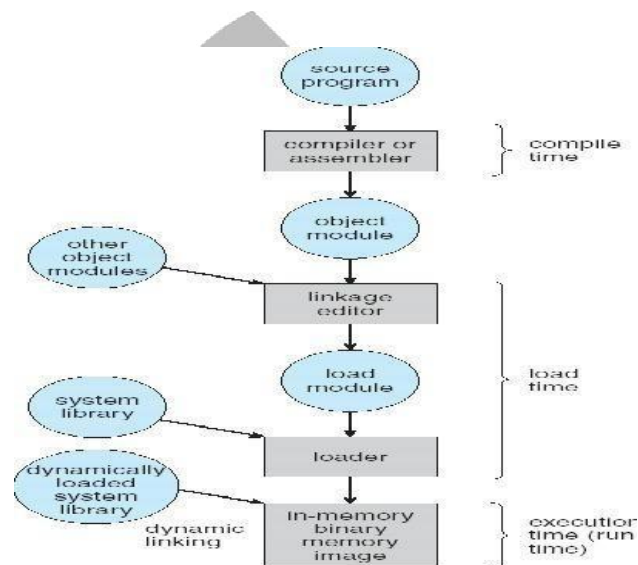
STORAGE MANAGEMENT

Memory Management: Background

- In general, to run a program, it must be brought into memory.
- Input queue – collection of processes on the disk that are waiting to be brought into memory to run the program.
- User programs go through several steps before being run
- Address binding: Mapping of instructions and data from one address to another address in memory.

Three different stages of binding:

1. Compile time: Must generate absolute code if memory location is known in prior.
2. Load time: Must generate relocatable code if memory location is not known at compile time
3. Execution time: Need hardware support for address maps (e.g., base and limit registers).



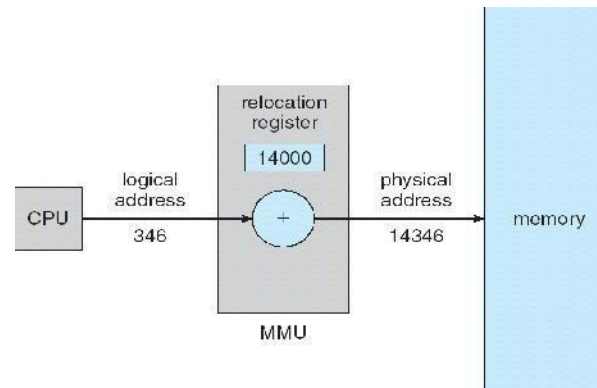
Logical vs. Physical Address Space

- **Logical address** – generated by the CPU; also referred to as “**virtual address**”
- **Physical address** – address seen by the memory unit.
- Logical and physical addresses are the **same** in compile-time and load-time address-binding schemes"
- Logical (virtual) and physical addresses **differ** in execution-time address-binding scheme"

Memory-Management Unit (MMU)

- It is a hardware device that maps virtual / Logical address to physical address
- In this scheme, the relocation register's value is added to Logical address generated by a user process.

- The user program deals with *logical* addresses; it never sees the *real* physical addresses
- Logical address range: 0 to max
- Physical address range: R+0 to R+max, where R—value in relocation register.



Dynamic Loading

- Through this, the routine is not loaded until it is called.
 - Better memory-space utilization; unused routine is never loaded
 - Useful when large amounts of code are needed to handle infrequently occurring cases
 - No special support from the operating system is required implemented through program design

Dynamic Linking

- Linking postponed until execution time & is particularly useful for libraries
- Small piece of code called stub, used to locate the appropriate memory-resident library routine or function.
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system needed to check if routine is in processes' memory address
- Shared libraries: Programs linked before the new library was installed will continue using the older library.

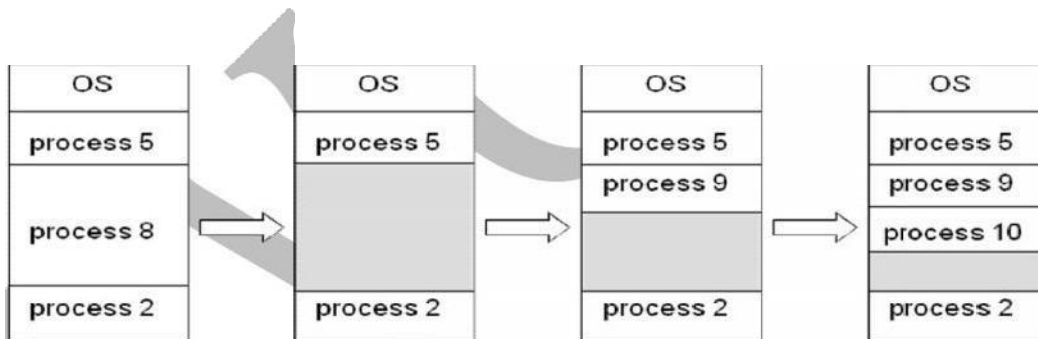
Swapping

- A process can be swapped temporarily out of memory to a backing store (SWAP OUT) and then brought back into memory for continued execution (SWAP IN).
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users & it must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- **Transfer time:** Major part of swap time is transfer time. Total transfer time is directly proportional to the amount of memory swapped.
- **Example:** Let us assume the user process is of size 1MB & the backing store is a standard hard disk with a transfer rate of 5MBPS.

Transfer time = 1000KB/5000KB per second
= 1/5 sec = 200ms

Contiguous Allocation

- Each process is contained in a single contiguous section of memory.
- There are two methods namely:
 - Fixed – Partition Method
 - Variable – Partition Method
- **Fixed – Partition Method :**
 - Divide memory into fixed size partitions, where each partition has exactly one process.
 - The drawback is memory space unused within a partition is wasted.(eg.when process size < partition size)
- **Variable-partition method:**
 - Divide memory into variable size partitions, depending upon the size of the incoming process.
 - When a process terminates, the partition becomes available for another process.
 - As processes complete and leave they create holes in the main memory.
 - **Hole** – block of available memory; holes of various size are scattered throughout memory.



Dynamic Storage- Allocation Problem:

How to satisfy a request of size n' from a list of free holes?

Solution:

- First-fit: Allocate the first hole that is big enough.
- Best-fit: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
- Worst-fit: Allocate the largest hole; must also search entire list. Produces the largest leftover hole.

NOTE: First-fit and best-fit are better than worst-fit in terms of speed and storage utilization

- **Fragmentation:**

- o **External Fragmentation** – This takes place when enough total memory space exists to satisfy a request, but it is not contiguous i.e, storage is fragmented into a large number of small holes scattered throughout the main memory.

- o **Internal Fragmentation** – Allocated memory may be slightly larger than requested memory.

Example: hole = 184 bytes

Process size = 182 bytes.

We are left with a hole of 2 bytes.

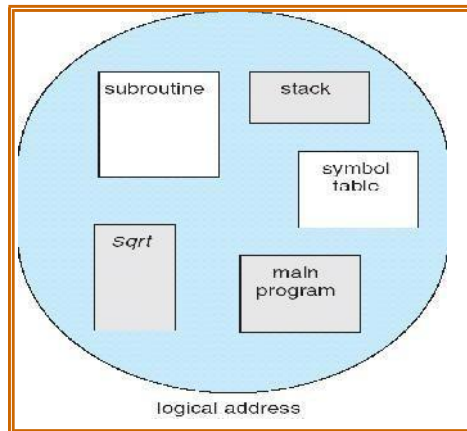
- o **Solutions**

1. **Coalescing:** Merge the adjacent holes together.
2. **Compaction:** Move all processes towards one end of memory, hole towards other end of memory, producing one large hole of available memory. This scheme is expensive as it can be done if relocation is dynamic and done at execution time.
3. Permit the logical address space of a process to be **non-contiguous**. This is achieved through two memory management schemes namely **paging** and **segmentation**.

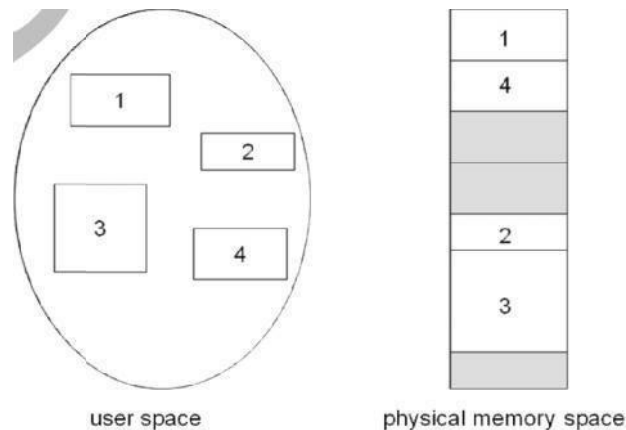
Segmentation

- o Memory-management scheme that supports user view of memory

- o A program is a collection of segments. A segment is a logical unit such as: Main program, Procedure, Function, Method, Object, Local variables, global variables, Common block, Stack, Symbol table, arrays

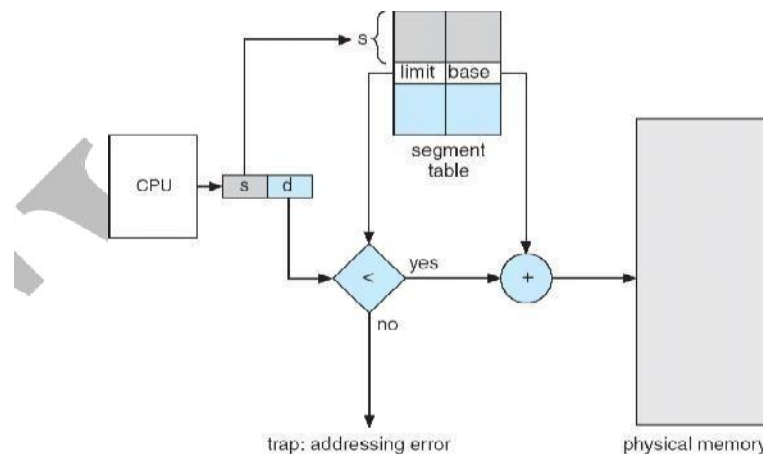


Logical View of Segmentation

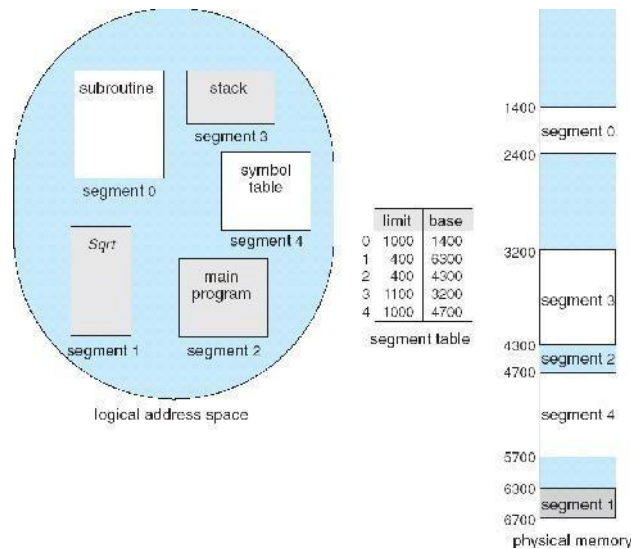


Segmentation Hardware

- o Logical address consists of a two tuple :
 <Segment-number, offset>
- o **Segment table** – maps two-dimensional physical addresses; each table entry has:
 - **Base** – contains the starting physical address where the segments reside in memory
 - **Limit** – specifies the length of the segment
- o **Segment-table base register (STBR)** points to the segment table's location in memory
- o **Segment-table length register (STLR)** indicates number of segments used by a program;
 Segment number= s is legal, if $s < STLR$
- o **Relocation.**
 - dynamic
 - by segment table
- o **Sharing.**
 - shared segments
 - same segment number
- o **Allocation.**
 - first fit/best fit
 - external fragmentation
- o **Protection:** With each entry in segment table associate:
 - validation bit = 0 □ illegal segment
 - read/write/execute privileges
- o Protection bits associated with segments; code sharing occurs at segment level
- o Since segments vary in length, memory allocation is a dynamic storage- allocation problem
- o A segmentation example is shown in the following diagram



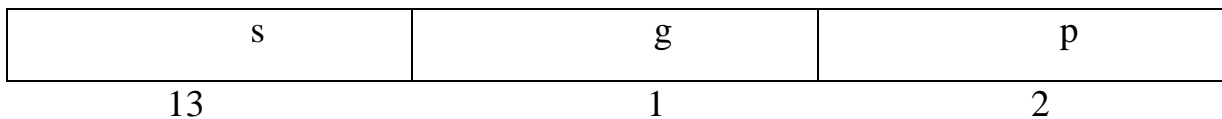
EXAMPLE:



- o Another advantage of segmentation involves the sharing of code or data.
- o Each process has a segment table associated with it, which the dispatcher uses to define the hardware segment table when this process is given the CPU.
- o Segments are shared when entries in the segment tables of two different processes point to the same physical location.

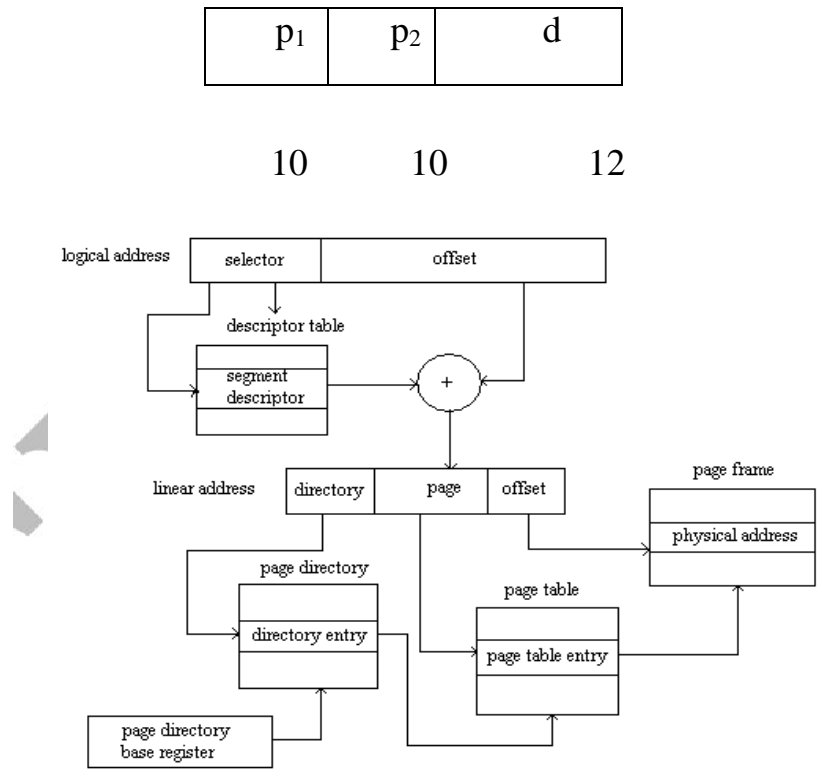
Segmentation with paging

- o The IBM OS/ 2.32 bit version is an operating system running on top of the Intel 386 architecture. The 386 uses segmentation with paging for memory management. The maximum number of segments per process is 16 KB, and each segment can be as large as 4 gigabytes.
 - o The local-address space of a process is divided into two partitions.
 - The first partition consists of up to 8 KB segments that are private to that process.
 - The second partition consists of up to 8KB segments that are shared among all the processes.
 - o Information about the first partition is kept in the **local descriptor table (LDT)**, information about the second partition is kept in the **global descriptor table (GDT)**.
 - o Each entry in the LDT and GDT consist of 8 bytes, with detailed information about a particular segment including the base location and length of the segment.
- The logical address is a pair (selector, offset) where the selector is a 16-bit number:



Where s designates the segment number, g indicates whether the segment is in the GDT or LDT, and p deals with protection. The offset is a 32-bit number specifying the location of the byte within the segment in question.

- o The base and limit information about the segment in question are used to generate a linear-address.
- o First, the limit is used to check for address validity. If the address is not valid, a memory fault is generated, resulting in a trap to the operating system. If it is valid, then the value of the offset is added to the value of the base, resulting in a 32-bit linear address. This address is then translated into a physical address.
- o The linear address is divided into a page number consisting of 20 bits, and a page offset consisting of 12 bits. Since we page the page table, the page number is further divided into a 10-bit page directory pointer and a 10-bit page table pointer. The logical address is as follows.



o To improve the efficiency of physical memory use. Intel 386 page tables can be swapped to disk. In this case, an invalid bit is used in the page directory entry to indicate whether the table to which the entry is pointing is in memory or on disk.

o If the table is on disk, the operating system can use the other 31 bits to specify the disk location of the table; the table then can be brought into memory on demand.

Paging

- It is a memory management scheme that permits the physical address space of a process to be noncontiguous.
- It avoids the considerable problem of fitting the varying size memory chunks on to the backing store.

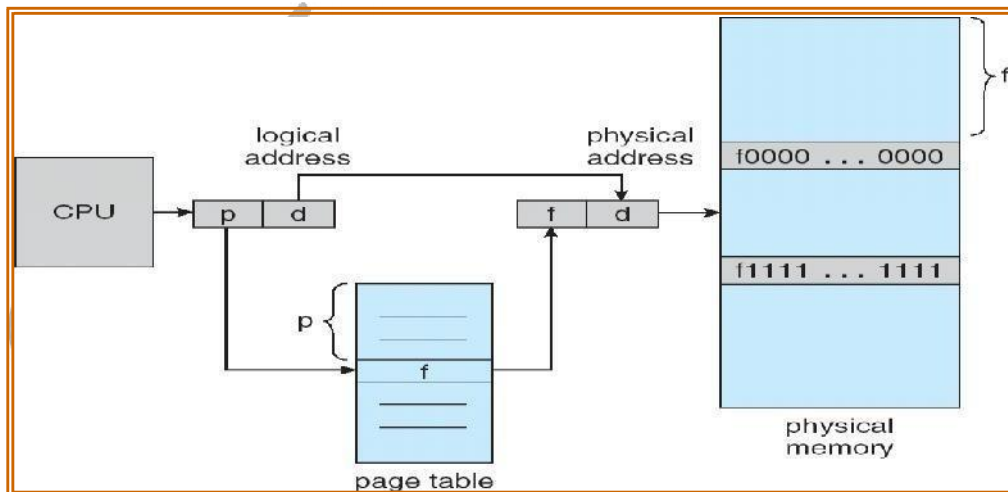
(i) Basic Method:

- o Divide logical memory into blocks of same size called “**pages**”.
- o Divide physical memory into fixed-sized blocks called “**frames**”
- o Page size is a power of 2, between 512 bytes and 16MB.

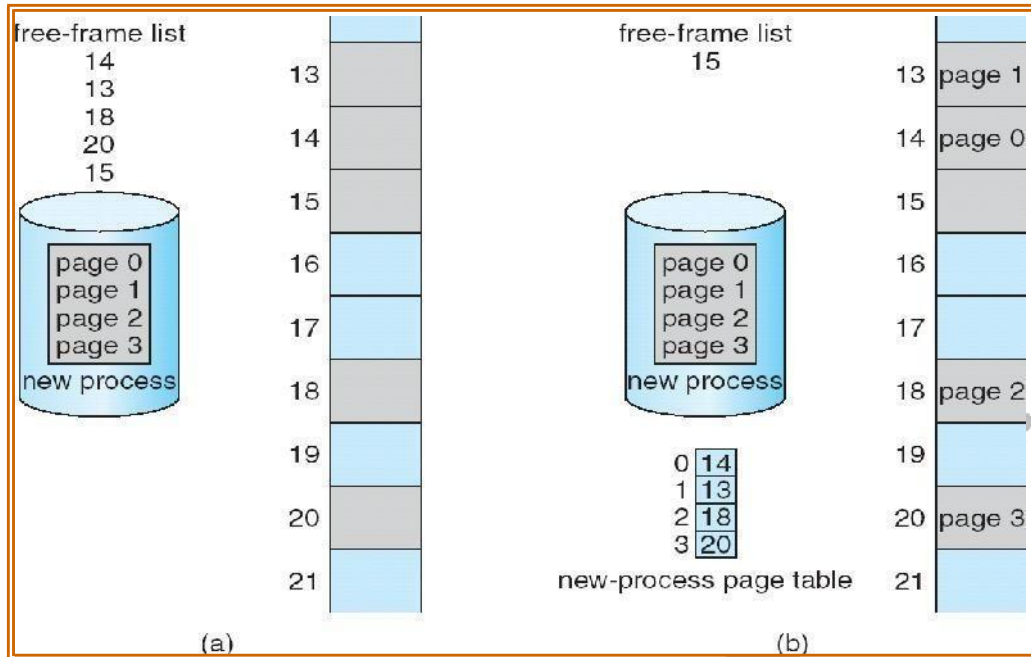
Address Translation Scheme

- o Address generated by CPU(logical address) is divided into:
 - **Page number (p)** – used as an index into a page table which contains base address of each page in physical memory
 - **Page offset (d)** –combined with base address to define the physical address i.e.,
$$\text{Physical address} = \text{base address} + \text{offset}$$

Paging Hardware



- o If n^{st} frames are available, they are allocated to this arriving process.
- o The 1st page of the process is loaded into one of the allocated frames & the frame number is put into the page table.
- o Repeat the above step for the next pages & so on.



(a) Before Allocation

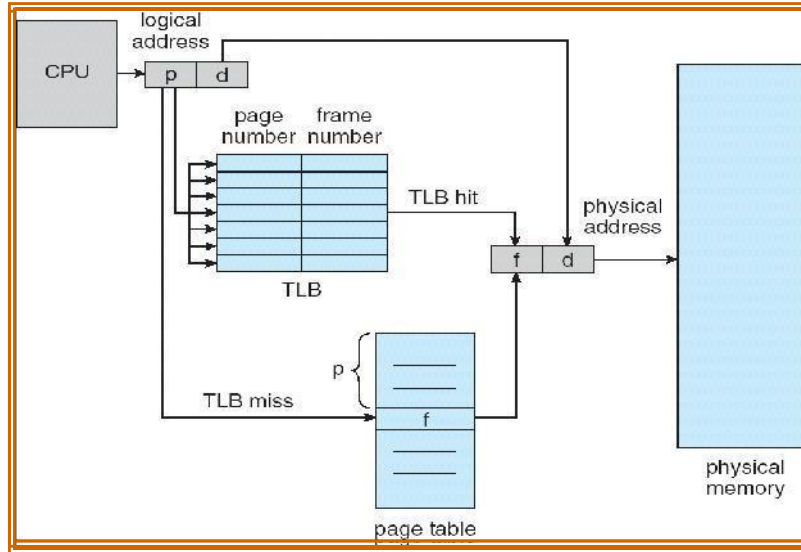
(b) After Allocation

Frame table: It is used to determine which frames are allocated, which frames are available, how many total frames are there, and so on. (ie) It contains all the information about the frames in the physical memory.

(ii) Hardware implementation of Page Table

- o This can be done in several ways :
 1. Using PTBR
 2. TLB
- o The simplest case is **Page-table base register (PTBR)**, is an index to point the pagetable.
- o **TLB (Translation Look-aside Buffer)**
 - It is a fast lookup hardware cache.
 - It contains the recently or frequently used page table entries.
 - It has two parts: Key (tag) & Value.
 - More expensive.

Paging Hardware with TLB



- When a logical address is generated by CPU, its page number is presented to TLB.
- **TLB hit:** If the page number is found, its frame number is immediately available & is used to access memory
- **TLB miss:** If the page number is not in the TLB, a memory reference to the page table must be made.
- **Hit ratio:** Percentage of times that a particular page is found in the TLB.
 - For example hit ratio is 80% means that the desired page number in the TLB is 80% of the time.
- **Effective Access Time:**
 - Assume hit ratio is 80%.
 - If it takes 20ns to search TLB & 100ns to access memory, then the memory access takes 120ns(TLB hit)
 - If we fail to find page no. in TLB (20ns), then we must 1st access memory for page table (100ns) & then access the desired byte in memory (100ns).

$$\text{Therefore Total} = 20 + 100 + 100$$

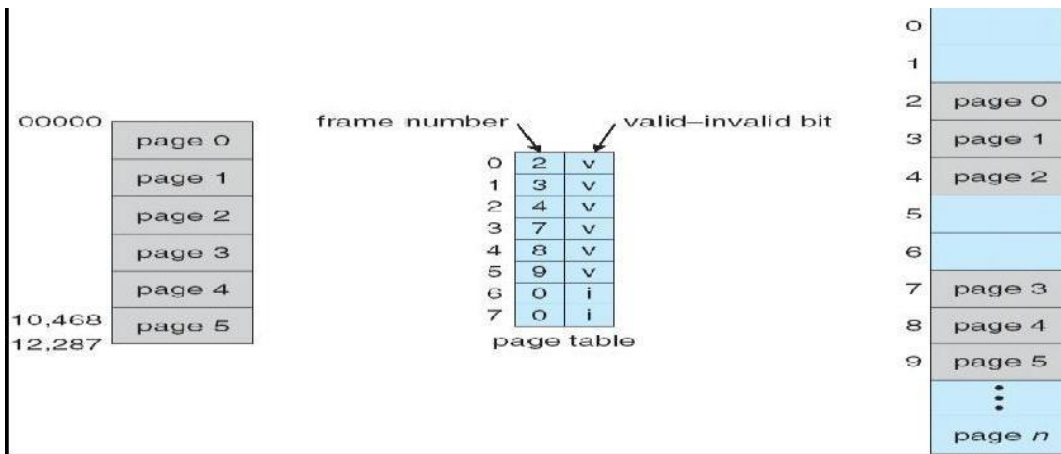
$$= 220 \text{ ns(TLB miss).}$$

$$\text{Then Effective Access Time (EAT)} = 0.80 \times (120 + 0.20) \times 220.$$

$$= 140 \text{ ns.}$$

(iii) Memory Protection

- Memory protection implemented by associating protection bit with each frame
- Valid-invalid bit attached to each entry in the page table:
 - **“valid (v)”** indicates that the associated page is in the process' logical address space, and is thus a legal page
 - **“invalid (i)”** indicates that the page is not in the process' logical address spaces



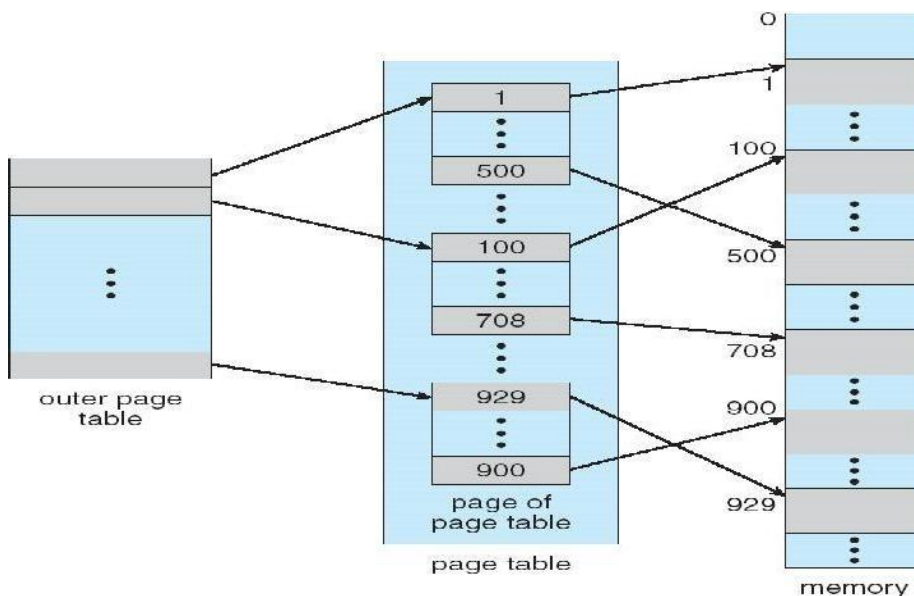
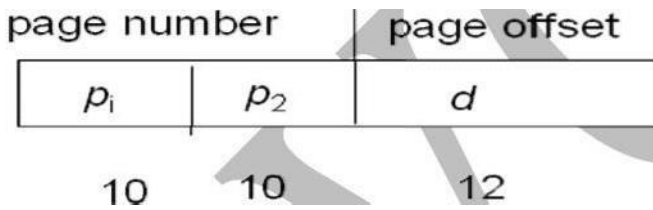
(iv) Structures of the Page Table

- a) Hierarchical Paging
- b) Hashed Page Tables
- c) Inverted Page Tables

a) Hierarchical Paging

o Break up the Page table into smaller pieces. Because if the page table is too large then it is difficult to search the page number.

Example: “Two-Level Paging “



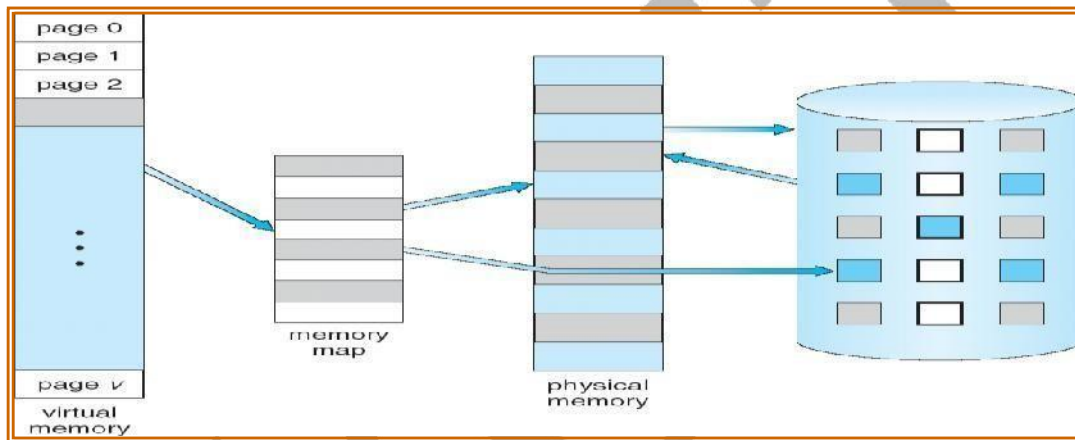
Virtual Memory

- o It is a technique that allows the execution of processes that may not be completely in main memory.
- o **Advantages:**
 - Allows the program that can be larger than the physical memory.
 - Separation of user logical memory from physical memory
 - Allows processes to easily share files & address space.
 - Allows for more efficient process creation.

Virtual memory can be implemented using

- Demand paging
- Demand segmentation

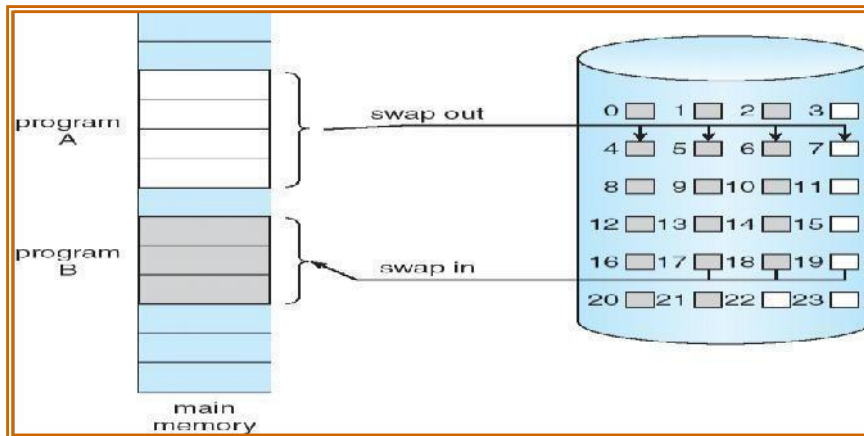
Virtual Memory That is Larger than Physical Memory



Demand Paging

- o It is similar to a paging system with swapping.
- o Demand Paging - Bring a page into memory only when it is needed
- o To execute a process, swap that entire process into memory. Rather than swapping the entire process into memory however, we use Lazy Swapper||
- o **Lazy Swapper** - Never swaps a page into memory unless that page will be needed.
- o **Advantages**
 - Less I/O needed
 - Less memory needed
 - Faster response
 - More users

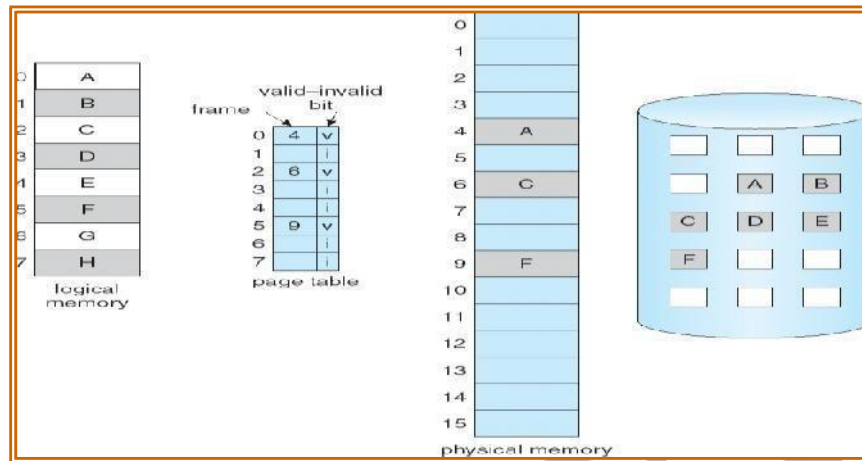
Transfer of a paged memory to contiguous disk space



Basic Concepts:

- o Instead of swapping in the whole processes, the pager brings only those necessary pages into memory. Thus,
 1. It avoids reading into memory pages that will not be used anyway.
 2. Reduce the swap time.
 3. Reduce the amount of physical memory needed.
- o To differentiate between those pages that are in memory & those that are on the disk we use the **Valid-Invalid bit**
- o A valid – invalid bit is associated with each page table entry.
- o Valid associated page is in memory.
- o In-Valid
 - invalid page
 - valid page but is currently on the disk

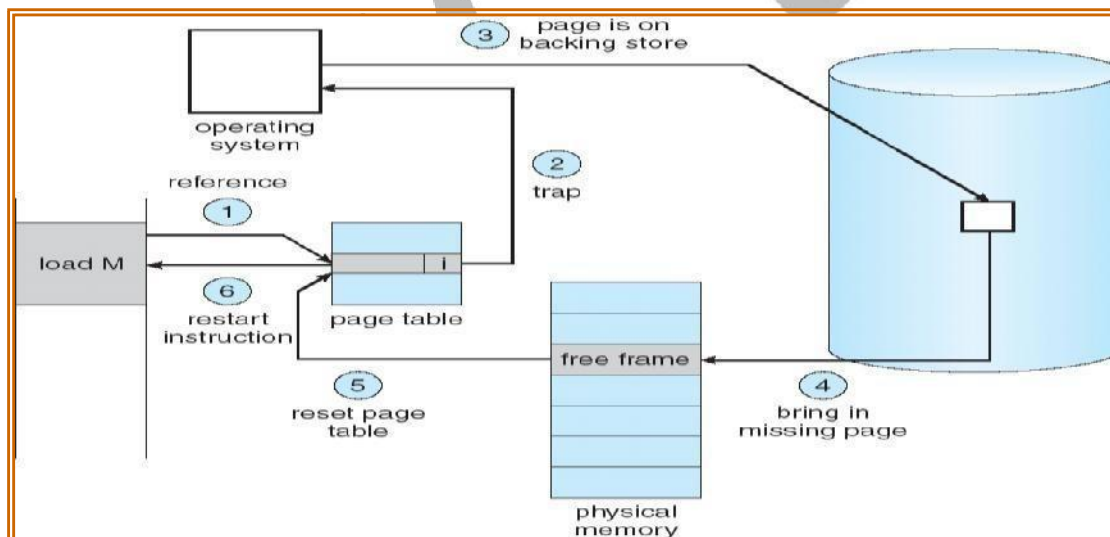
Page table when some pages are not in main memory



Page Fault

o Access to a page marked invalid causes a page fault trap.

Steps in Handling a Page Fault



1. Determine whether the reference is a valid or invalid memory access
2. a) If the reference is invalid then terminate the process.
b) If the reference is valid then the page has not been yet brought into main memory.
3. Find a free frame.
4. Read the desired page into the newly allocated frame.
5. Reset the page table to indicate that the page is now in memory.
6. Restart the instruction that was interrupted .

Pure demand paging

- o Never bring a page into memory until it is required.
- o We could start a process with no pages in memory.
- o When the OS sets the instruction pointer to the 1st instruction of the process, which is on the non-memory resident page, then the process immediately faults for the page.
- o After this page is brought into the memory, the process continues to execute, faulting as necessary until every page that it needs is in memory.

Performance of demand paging

- o Let p be the probability of a page fault $0 \leq p \leq 1$
- o Effective Access Time (EAT)
$$EAT = (1 - p) \times m_a + p \times \text{page fault time.}$$

Where m_a = memory access, p = Probability of page fault ($0 \leq p \leq 1$)

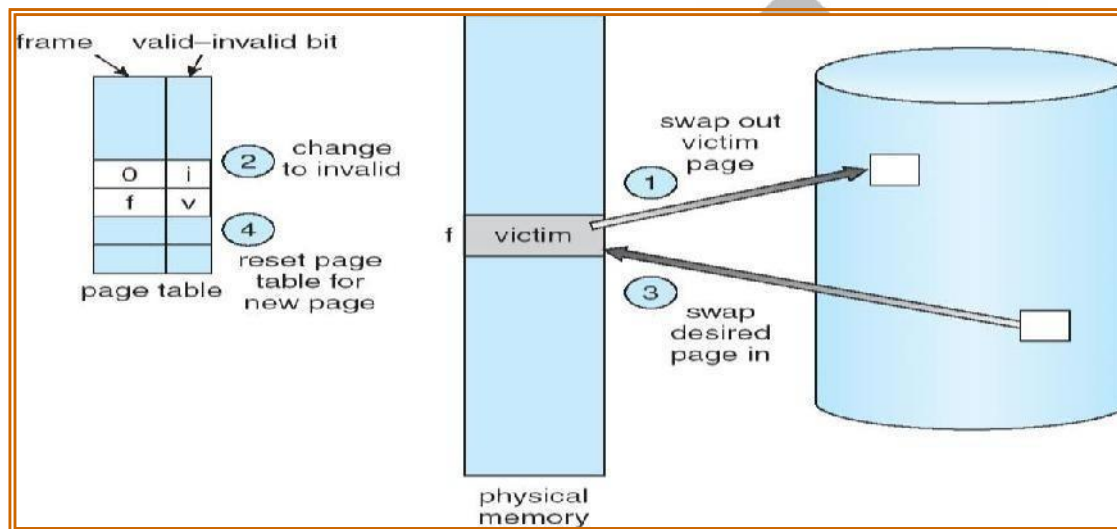
- o The memory access time denoted m_a is in the range 10 to 200 ns.
- o If there are no page faults then $EAT = m_a$.
- o To compute effective access time, we must know how much time is needed to service a page fault.
- o A page fault causes the following sequence to occur:
 1. Trap to the OS
 2. Save the user registers and process state.
 3. Determine that the interrupt was a page fault.
 4. Check whether the reference was legal and find the location of page on disk.
 5. Read the page from disk to free frame.
 - a. Wait in a queue until read request is serviced.
 - b. Wait for seek time and latency time.
 - c. Transfer the page from disk to free frame.
 6. While waiting, allocate CPU to some other user.
 7. Interrupt from disk.
 8. Save registers and process state for other users.
 9. Determine that the interrupt was from disk.
 7. Reset the page table to indicate that the page is now in memory.
 8. Wait for CPU to be allocated to this process again.
 9. Restart the instruction that was interrupted.

Page Replacement

- o If no frames are free, we could find one that is not currently being used & free it.
- o We can free a frame by writing its contents to swap space & changing the page table to indicate that the page is no longer in memory.
- o Then we can use that freed frame to hold the page for which the process faulted.

Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame
 - If there is a free frame , then use it.
 - If there is no free frame, use a page replacement algorithm to select a **victim** frame
 - Write the victim page to the disk, change the page & frame tables accordingly.
3. Read the desired page into the (new) free frame. Update the page and frame tables.
4. Restart the process



Page Replacement Algorithms

1. FIFO Page Replacement
2. Optimal Page Replacement
3. LRU Page Replacement
4. LRU Approximation Page Replacement
5. Counting-Based Page Replacement

(a) FIFO page replacement algorithm

- o **Replace the oldest page.**
- o This algorithm associates with each page ,the time when that page was brought in.

Example:

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

No.of available frames = 3 (3 pages can be in memory at a time per process)

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2	2	2	4	4	4	0	0	0	7	7	7
	0	0	0	3	3	3	2	2	2	1	1	1	0	0
		1	1	1	0	0	0	3	3	3	2	2	2	1

page frames

No. of page faults = 15

Drawback:

- o FIFO page replacement algorithm's performance is not always good.
- o To illustrate this, consider the following example:

Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- o If No. of available frames = 3 then the no. of page faults = 9
- o If No. of available frames = 4 then the no. of page faults = 10
- o Here the no. of page faults increases when the no. of frames increases. This is called as **Belady's Anomaly**.

(b) Optimal page replacement algorithm

- o **Replace the page that will not be used for the longest period of time.**

Example:

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

No. of available frames = 3

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

7	7	7	2	2	2	2	2	7
	0	0	0	0	4	0	0	0
		1	1	3	3	3	1	1

page frames

No. of page faults = 9

Drawback:

o It is difficult to implement as it requires future knowledge of the reference string.

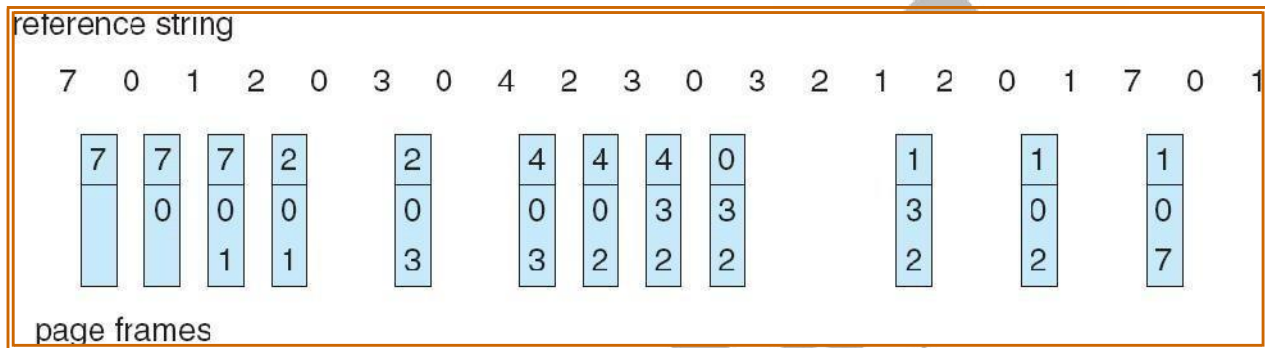
(c) LRU(Least Recently Used) page replacement algorithm

o **Replace the page that has not been used for the longest period of time.**

Example:

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

No.of available frames = 3



No. of page faults = 12

o LRU page replacement can be implemented using

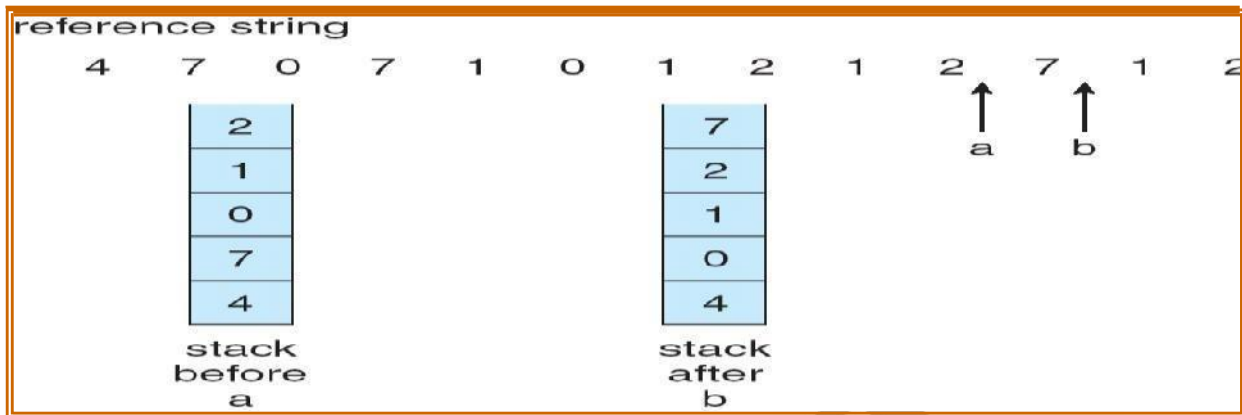
1. Counters

- Every page table entry has a time-of-use field and a clock or counter is associated with the CPU.
- The counter or clock is incremented for every memory reference.
- Each time a page is referenced , copy the counter into the time-of-use field.
- When a page needs to be replaced, replace the page with the smallest counter value.

2. Stack

- Keep a stack of page numbers
- Whenever a page is referenced, remove the page from the stack and put it on top of the stack.
- When a page needs to be replaced, replace the page that is at the bottom of the stack.(LRU page)

Use of A Stack to Record The Most Recent Page References



(d) LRU Approximation Page Replacement

- o Reference bit
 - With each page associate a reference bit, initially set to 0
 - When page is referenced, the bit is set to 1
- o When a page needs to be replaced, replace the page whose reference bit is 0
- o The order of use is not known , but we know which pages were used and which were not used.

(i) Additional Reference Bits Algorithm

- o Keep an 8-bit byte for each page in a table in memory.
- o At regular intervals , a timer interrupt transfers control to OS.
- o The OS shifts reference bit for each page into higher- order bit shifting the other bits right 1 bit and discarding the lower-order bit.

Example:

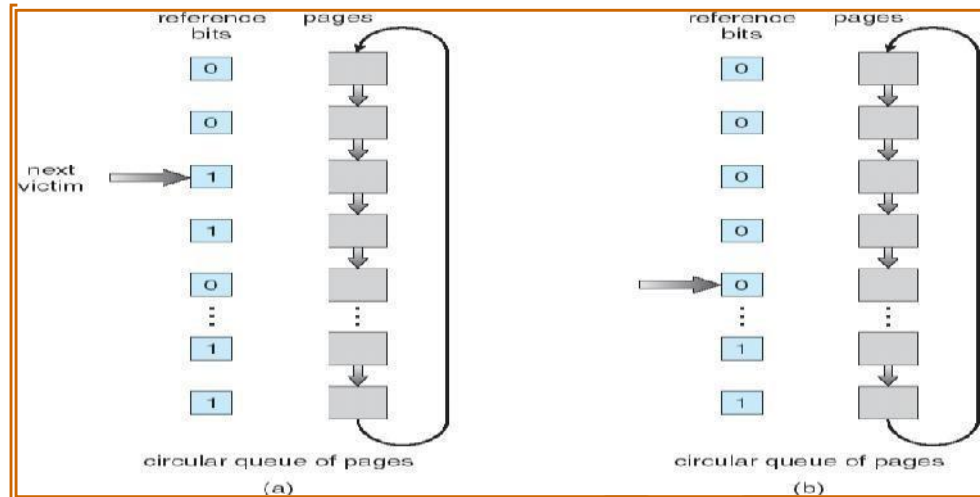
oIf reference bit is 00000000 then the page has not been used for 8 time periods.

oIf reference bit is 11111111 then the page has been used atleast once each time period.

oIf the reference bit of page 1 is 11000100 and page 2 is 01110111 then page 2 is the LRU page.

(ii) Second Chance Algorithm

- oBasic algorithm is FIFO
- oWhen a page has been selected , check its reference bit.
 - If 0 proceed to replace the page
 - If 1 give the page a second chance and move on to the next FIFO page.
 - When a page gets a second chance, its reference bit is cleared and arrival time is reset to current time.
 - Hence a second chance page will not be replaced until all other pages are replaced.



(iii) **Enhanced Second Chance Algorithm** o Consider both reference bit and modify bit o There are four possible classes

1. (0,0) – neither recently used nor modified □ Best page to replace
2. (0,1) – not recently used but modified □ page has to be written out before replacement.
3. (1,0) - recently used but not modified □ page may be used again
4. (1,1) – recently used and modified □ page may be used again and page has to be written to disk

(e) Counting-Based Page Replacement

- o Keep a counter of the number of references that have been made to each page
 1. **Least Frequently Used (LFU)Algorithm:** replaces page with smallest count
 2. **Most Frequently Used (MFU)Algorithm:** replaces page with largest count
 - It is based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Page Buffering Algorithm

- o These are used along with page replacement algorithms to improve their performance

Technique 1:

- o A pool of free frames is kept.
- o When a page fault occurs, choose a victim frame as before.
- o Read the desired page into a free frame from the pool
- o The victim frame is written onto the disk and then returned to the pool of free frames.

Technique 2:

- o Maintain a list of modified pages.
- o Whenever the paging device is idles, a modified is selected and written to disk and its modify bit is reset.

Technique 3:

- o A pool of free frames is kept.
- o Remember which page was in each frame.
- o If frame contents are not modified then the old page can be reused directly from the free frame pool when needed

Allocation of Frames

- o There are two major allocation schemes
 - Equal Allocation
 - Proportional Allocation
- o **Equal allocation**
 - If there are n processes and m frames then allocate m/n frames to each process.
 - **Example:** If there are 5 processes and 100 frames, give each process 20 frames.

- o **Proportional allocation**

- Allocate according to the size of process

Let s_i be the size of process i.

Let m be the total no. of frames

Then $S = \sum s_i$

$a_i = s_i / S * m$

where a_i is the no.of frames allocated to process i.

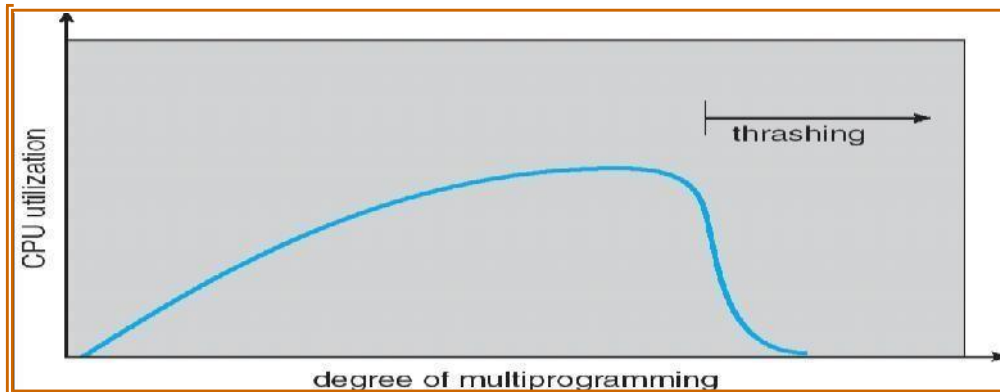
Global vs. Local Replacement

- o **Global replacement** – each process selects a replacement frame from the set of all frames; one process can take a frame from another.
- o **Local replacement** – each process selects from only its own set of allocated frames.

Thrashing

- o High paging activity is called **thrashing**.
- o If a process does not have enough pages, the page-fault rate is very high.
 - This leads to:
 - low CPU utilization
 - operating system thinks that it needs to increase the degree of multiprogramming
 - another process is added to the system
- o When the CPU utilization is low, the OS increases the degree of multiprogramming.
- o If global replacement is used then as processes enter the main memory they tend to steal frames belonging to other processes.
- o Eventually all processes will not have enough frames and hence the page fault rate becomes very

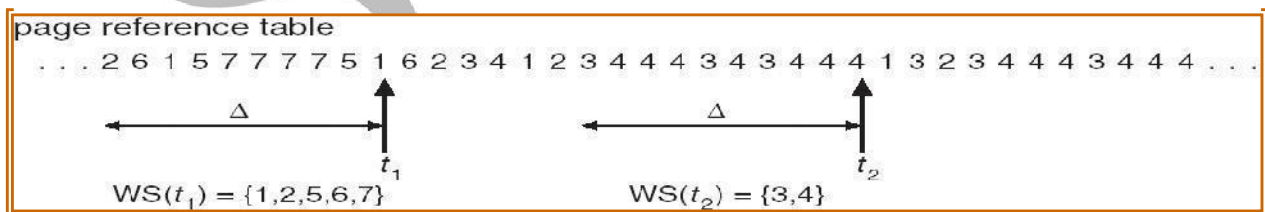
- high.
- o Thus swapping in and swapping out of pages only takes place.
- o This is the cause of thrashing.



- o To **limit thrashing**, we can use a **local replacement** algorithm.
- o To prevent thrashing, there are two methods namely ,
 - Working Set Strategy
 - Page Fault Frequency

1. Working-Set Strategy

- o It is based on the assumption of the model of locality.
- o Locality is defined as the set of pages actively used together.
- o Working set is the set of pages in the most recent Δ page references
- o Δ is the working set window.
 - if Δ too small , it will not encompass entire locality
 - if Δ too large ,it will encompass several localities
 - if $\Delta = \infty$ it will encompass entire program
- o $D = \sum WSS_i$
 - Where WSS_i is the working set size for process i.
 - D is the total demand of frames
 - o if $D > m$ then Thrashing will occur.



2. Page-Fault Frequency Scheme

- o If actual rate too low, process loses frame
- o If actual rate too high, process gains frame



Other Issues

o Prepaging

- To reduce the large number of page faults that occurs at process startup
- Prepage all or some of the pages a process will need, before they are referenced
- But if prepagged pages are unused, I/O and memory are wasted

o Page Size

Page size selection must take into consideration:

- o fragmentation
- o table size
- o I/O overhead
- o locality

o TLB Reach

- TLB Reach - The amount of memory accessible from the TLB
- $TLB\ Reach = (TLB\ Size) \times (Page\ Size)$
- Ideally, the working set of each process is stored in the TLB. Otherwise there is a high degree of page faults.
- Increase the Page Size. This may lead to an increase in fragmentation as not all applications require a large page size
- Provide Multiple Page Sizes. This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation.

o I/O interlock

- Pages must sometimes be locked into memory
- Consider I/O. Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm.

Allocating Kernel Memory

When a process running in user mode requests additional memory, pages are allocated from the list of free page frames maintained by the kernel. This list is typically populated using a page-replacement algorithm such as those discussed in Section 9.4 and most likely contains free pages scattered throughout physical memory, as explained earlier. Remember, too, that if a user process requests a single byte of memory, internal fragmentation will result, as the process will be granted an entire page frame.

Kernel memory is often allocated from a free-memory pool different from the list used to satisfy ordinary user-mode processes. There are two primary reasons for this:

1. The kernel requests memory for data structures of varying sizes, some of which are less than a page in size. As a result, the kernel must use memory conservatively and attempt to minimize waste due to fragmentation. This is especially important because many operating systems do not subject kernel code or data to the paging system.

2. Pages allocated to user-mode processes do not necessarily have to be in contiguous physical memory. However, certain hardware devices interact directly with physical memory—without the benefit of a virtual memory interface—and consequently may require memory residing in physically contiguous pages.

Buddy System

The buddy system allocates memory from a fixed -size segment consisting of physically contiguous pages. Memory is allocated from this segment using a **power-of-2 allocator**, which satisfies requests in units sized as a power of 2 (4KB,8KB,16KB, and so forth). A request in units not appropriately sized is rounded up to the next highest power of 2. For example, a request for 11 KB is satisfied with a 16K segment

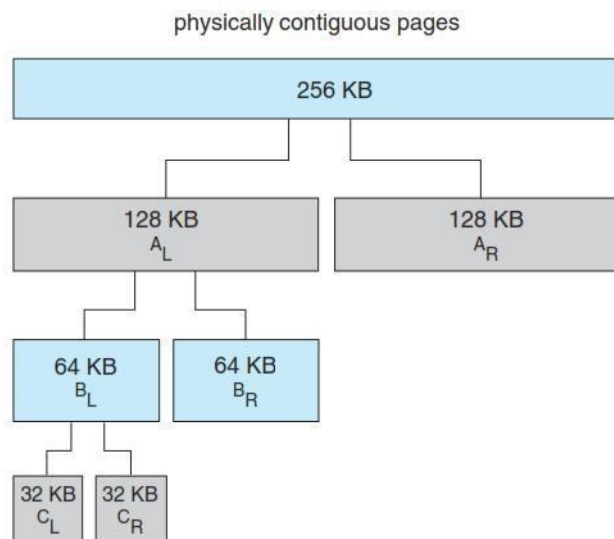


Figure 9.26 Buddy system allocation.

OS Examples

Windows

Windows implements virtual memory using demand paging with clustering. Clustering handles page faults by bringing in not only the faulting page but also several pages following the faulting page. When a process is first created, it is assigned a working-set minimum and maximum. The **working-set minimum** is the minimum number of pages the process is guaranteed to have in memory.

If sufficient memory is available, a process may be assigned as many pages as its **working-set maximum**. (In some circumstances, a process may be allowed to exceed its working-set maximum.) The virtual memory manager maintains a list of free page frames. Associated with this list is a threshold value that is used to indicate whether sufficient free memory is available. If a page fault occurs for a process that is below its working-set maximum, the virtual memory manager allocates a page from this list of free pages. If a process that is at its working-set maximum incurs a page fault, it must select a page for replacement using a local LRU page-replacement policy.

Solaris

In Solaris, when a thread incurs a page fault, the kernel assigns a page to the faulting thread from the list of free pages it maintains. Therefore, it is imperative that the kernel keep a sufficient amount of free memory available. Associated with this list of free pages is a parameter—**lotsfree**—that represents a threshold to begin paging. The **lotsfree** parameter is typically set to 1/64 the size of the physical memory. Four times per second, the kernel checks whether the amount of free memory is less than **lotsfree**. If the number of free pages falls below **lotsfree**, a process known as a **pageout** starts up. The pageout process is similar to the second



Figure 9.29 Solaris page scanner.

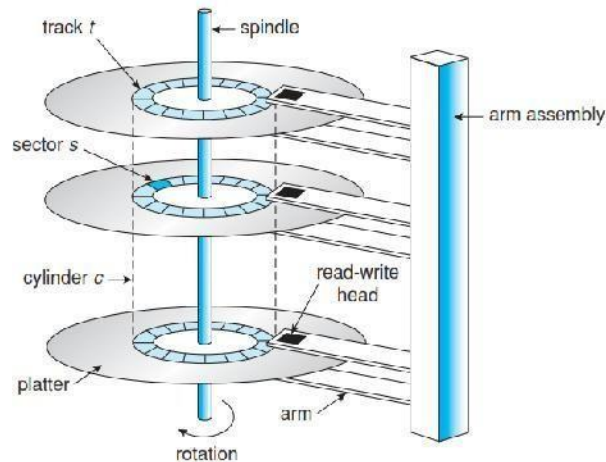
CS6401- Operating System

UNIT-IV

I/O SYSTEMS

Magnetic Disks

Magnetic disks provide the bulk of secondary storage for modern computer systems. Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 3.5 inches. The two surfaces of a platter are covered with a magnetic material. We store information by recording it magnetically on the platters



A read–write head “flies” just above each surface of every platter. The heads are attached to a **disk arm** that moves all the heads as a unit. The surface of a platter is logically divided into circular **tracks**, which are subdivided into **sectors**. The set of tracks that are at one arm position makes up a **cylinder**. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors. The storage capacity of common disk drives is measured in gigabytes.

A disk drive is attached to a computer by a set of wires called an **I/O bus**. Several kinds of buses are available, including **advanced technology attachment (ATA)**, **serial ATA (SATA)**, **eSATA**, **universal serial bus (USB)**, and **fibre channel (FC)**. The data transfers on a bus are carried out by special electronic processors called **controllers**. The **host controller** is the controller at the computer end of the bus. A **disk controller** is built into each disk drive. To perform a disk I/O operation, the computer places a command into the host controller, typically using memory-mapped I/O ports.

The host controller then sends the command via messages to the disk controller, and the disk controller operates the disk-drive hardware to carry out the command. Disk controllers usually have a built-in cache. Data transfer at the disk drive happens between the cache and the disk surface, and data transfer to the host, at fast electronic speeds, occurs between the cache and the host controller.

Solid-State Disks

Sometimes old technologies are used in new ways as economics change or the technologies evolve. An example is the growing importance of **solid-state disks**, or **SSDs**. Simply described, an SSD is nonvolatile memory that is used like a hard drive. There are many variations of this technology, from DRAM with a battery to allow it to maintain its state in a power failure through flash-memory technologies like single-level cell (SLC) and multilevel cell (MLC) chips.

SSDs have the same characteristics as traditional hard disks but can be more reliable because they have no moving parts and faster because they have no seek time or latency. In addition, they consume less power. However, they are more expensive per megabyte than traditional hard disks, have less capacity than the larger hard disks, and may have shorter life spans than hard disks, so their uses are somewhat limited. One use for SSDs is in storage arrays, where they hold file-system metadata that require high performance. SSDs are also used in some laptop computers to make them smaller, faster, and more energy-efficient. Because SSDs can be much faster than magnetic disk drives, standard bus interfaces can cause a major limit on throughput.

Magnetic Tapes

Magnetic tape was used as an early secondary-storage medium. Although it is relatively permanent and can hold large quantities of data, its access time is slow compared with that of main memory and magnetic disk. In addition, random access to magnetic tape is about a thousand times slower than random access to magnetic disk, so tapes are not very useful for secondary storage. Tapes are used mainly for backup, for storage of infrequently used information, and as a medium for transferring information from one system to another.

A tape is kept in a spool and is wound or rewound past a read–write head. Moving to the correct spot on a tape can take minutes, but once positioned, tape drives can write data at speeds comparable to disk drives. Tape capacities vary greatly, depending on the particular kind of tape drive, with current capacities exceeding several terabytes. Some tapes have built-in compression that can more than double the effective storage. Tapes and their drivers are usually categorized by width, including 4, 8, and 19 millimetres and 1/4 and 1/2 inch. Some are named according to technology, such as LTO-5 and SDLT.

Disk Scheduling and Management

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives,

1. A fast access time and
 2. High disk bandwidth.
- The **access time** has two major components;
 - The **seek time** is the time for the disk arm to move the heads to the cylinder containing the desired sector.
 - The **rotational latency** is the additional time waiting for the disk to rotate the desired sector to the disk head.
 - The disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

One of the responsibilities of the operating system is to use the hardware efficiently.

For the disk drives,

1. A fast access time and
2. High disk bandwidth.

□ The **access time** has two major components;

✓ The **seek time** is the time for the disk arm to move the heads to the cylinder containing the desired sector.

✓ The **rotational latency** is the additional time waiting for the disk to rotate the desired sector to the disk head.

□ The disk **bandwidth** is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer.

We can improve both the access time and the bandwidth by disk scheduling.

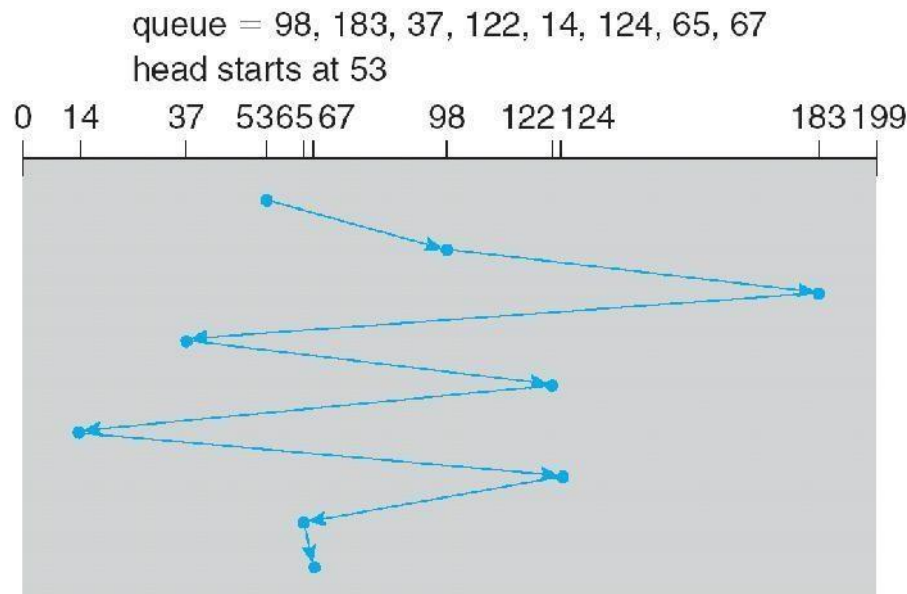
Disk scheduling: Servicing of disk I/O requests in a good order.

1. FCFS Scheduling:

The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. Consider, for example, a disk queue with requests for I/O to blocks on cylinders

I/O to blocks on cylinders

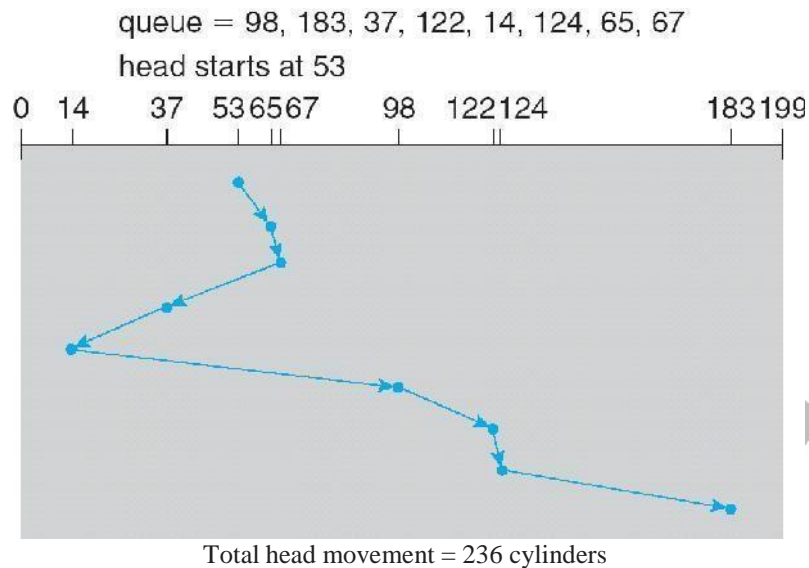
98, 183, 37, 122, 14, 124, 65, 67,



If the disk head is initially at cylinder 53, it will first move from 53 to 98, then to 183, 37, 122, 14, 124, 65, and finally to 67, for a **total head movement of 640 cylinders**. The wild swing from 122 to 14 and then back to 124 illustrates the problem with this schedule. If the requests for cylinders 37 and 14 could be serviced together, before or after the requests for 122 and 124, the total head movement could be decreased substantially, and performance could be thereby improved.

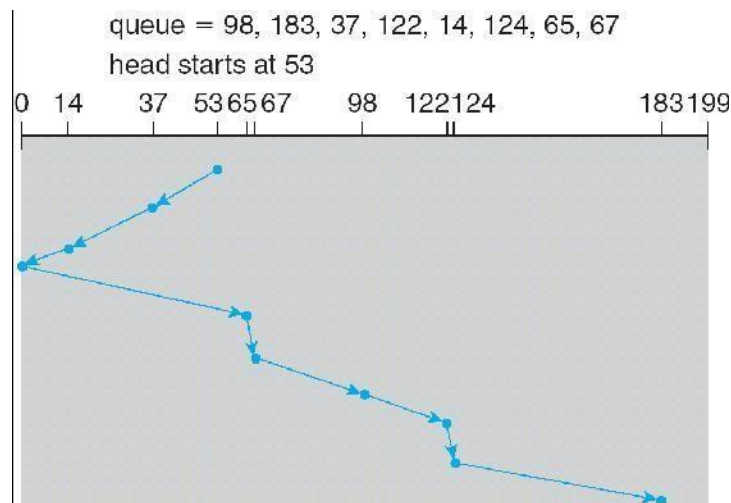
2. SSTF (shortest-seek-time-first) Scheduling

Service all the requests close to the current head position, before moving the head far away to service other requests. That is selects the request with the minimum seek time from the current head position.



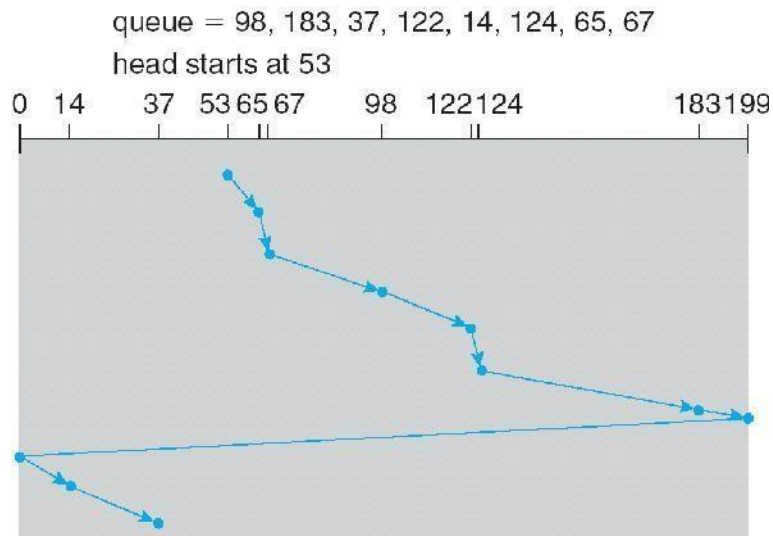
3. SCAN Scheduling

The disk head starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk.



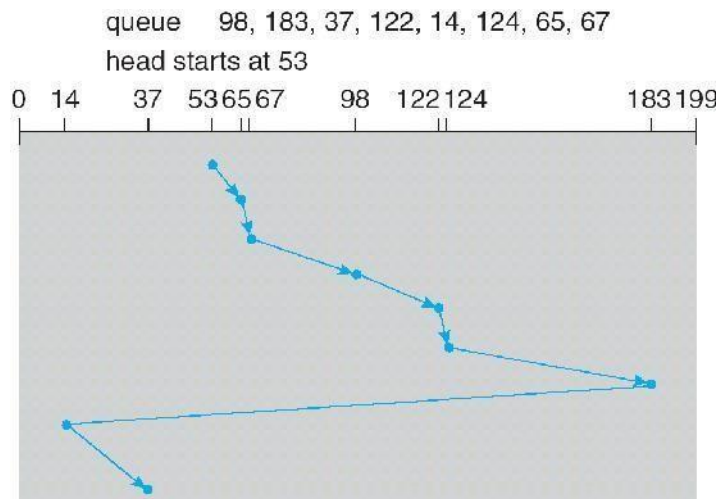
4. C-SCAN Scheduling

Variant of SCAN designed to provide a more uniform wait time. It moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.



5. LOOK Scheduling

Both SCAN and C-SCAN move the disk arm across the full width of the disk. In this, the arm goes only as far as the final request in each direction. Then, it reverses direction immediately, without going all the way to the end of the disk.



Disk Management

1. Disk Formatting:

Before a disk can store data, the sector is divided into various partitions. This process is called low-level formatting or physical formatting. It fills the disk with a special data structure for each sector.

The data structure for a sector consists of

- ✓ Header,
- ✓ Data area (usually 512 bytes in size), and
- ✓ Trailer.

The header and trailer contain information used by the disk controller, such as a sector number and an **error-correcting code (ECC)**.

This formatting enables the manufacturer to

1. Test the disk and
2. To initialize the mapping from logical block numbers

To use a disk to hold files, the operating system still needs to record its own data structures on the disk. It does so in two steps.

- (a) The first step is **Partition** the disk into one or more groups of cylinders. Among the partitions, one partition can hold a copy of the OS's executable code, while another holds userfiles.
- (b) The second step is **logical formatting**. The operating system stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space and an initial empty directory.

2. **Boot Block:**

For a computer to start running—for instance, when it is powered up or rebooted—it needs to have an initial program to run. This initial program is called bootstrap program & it should be simple. It initializes all aspects of the system, from CPU registers to device controllers and the contents of main memory, and then starts the operating system.

To do its job, the bootstrap program

1. Finds the operating system kernel on disk,
2. Loads that kernel into memory, and
3. Jumps to an initial address to begin the operating-system execution. The bootstrap is stored in read-only memory (**ROM**).

Advantages:

1. ROM needs no initialization.
2. It is at a fixed location that the processor can start executing when powered up or reset.
3. It cannot be infected by a computer virus. Since, ROM is read only.

The full bootstrap program is stored in a partition called the **boot blocks**, at a fixed location on the disk. A disk that has a boot partition is called a **boot disk or system disk**.

The code in the boot ROM instructs the disk controller to read the boot blocks into memory and then starts executing that code.

Bootstrap loader: load the entire operating system from a non-fixed location on disk, and to start the operating system running.

3. **Bad Blocks:**

The disk with defected sector is called as bad block.

Depending on the disk and controller in use, these blocks are handled in a variety of ways;

Method 1: “Handled manually”

If blocks go bad during normal operation, a **special program** must be run manually to search for the bad blocks and to lock them away as before. Data that resided on the bad blocks usually are lost.

Method 2: “sector sparing or forwarding”

The controller maintains a list of bad blocks on the disk. Then the controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as sector sparing or forwarding.

A typical bad-sector transaction might be as follows:

1. The operating system tries to read logical block 87.
2. The controller calculates the ECC and finds that the sector is bad.
3. It reports this finding to the operating system.
4. The next time that the system is rebooted, a special command is run to tell the controller to replace the bad sector with a spare.
5. After that, whenever the system requests logical block 87, the request is translated into the replacement sector's address by the controller.

Method 3: “sector slipping”

For an example, suppose that logical block 17 becomes defective, and the first available spare follows sector 202. Then, sector slipping would remap all the sectors from 17 to 202, moving them all down one spot. That is, sector 202 would be copied into the spare, then sector 201 into 202, and then 200 into 201, and so on, until sector 18 is copied into sector 19. Slipping the sectors in this way frees up the space of sector 18, so sector 17 can be mapped to it.

File System Storage-File Concepts**File Concept**

A file is a named collection of related information that is recorded on secondary storage.

- From a user’s perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.

Examples of files:

- A text file is a sequence of characters organized into lines (and possibly pages). A source file is a sequence of subroutines and functions, each of which is further organized as declarations followed by executable statements. An object file is a sequence of bytes organized into blocks understandable by the system’s linker.

An executable file is a series of code sections that the loader can bring into memory and execute.

File Attributes

- **Name:** The symbolic file name is the only information kept in human readable form.
- **Identifier:** This unique tag, usually a number identifies the file within the file system. It is thenon-human readable name for the file.
- **Type:** This information is needed for those systems that support different types.

- **Location:** This information is a pointer to a device and to the location of the file on that device.
- **Size:** The current size of the file (in bytes, words or blocks) and possibly the maximum allowed size are included in this attribute.
- **Protection:** Access-control information determines who can do reading, writing, executing and so on.
- **Time, date and user identification:** This information may be kept for creation, last modification and last use. These data can be useful for protection, security and usage monitoring.

File Operations

- Creating a file
- Writing a file
- Reading a file
- Repositioning within a file
- Deleting a file
- Truncating a file

Access Methods

1. Sequential Access

- The simplest access method is sequential access. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editors and compilers usually access files in this fashion.

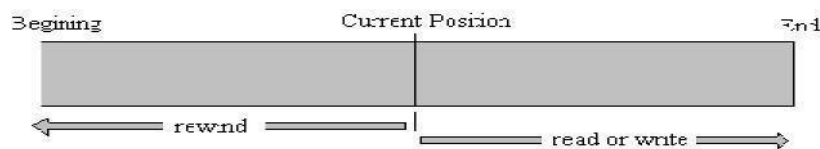


Fig 4.10 Sequential-access file

The bulk of the operations on a file is reads and writes. A read operation reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, a write appends to the end of the file and advances to the end of the newly written material (the new end of file). Such a file can be reset to the beginning and, on some systems, a program may be able to skip forward or back ward n records, for some integer n —perhaps only for $n=1$. Sequential access is based on a tape model of a file, and works as well on sequential-access devices as it does on random – access ones.

2. Direct Access

Another method is direct access (or relative access). A file is made up of fixed length logical records that allow programs to read and write records rapidly in no particular order. The direct- access methods is based on a disk model of a file, since disks allow random access to any file block.

For direct access, the file is viewed as a numbered sequence of blocks or records. A direct-access file allows arbitrary blocks to be read or written. Thus, we may read block 14, then read block 53, and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file.

Direct – access files are of great use for immediate access to large amounts of information. Database is often of this type. When a query concerning a particular subject arrives, we compute which block contains the answer, and then read that block directly to provide the desired information.

Directory and Disk Structure

There are five directory structures. They are

1. Single-level directory
2. Two-level directory
3. Tree-Structured directory
4. Acyclic Graph directory
5. General Graph directory

1. Single – Level Directory

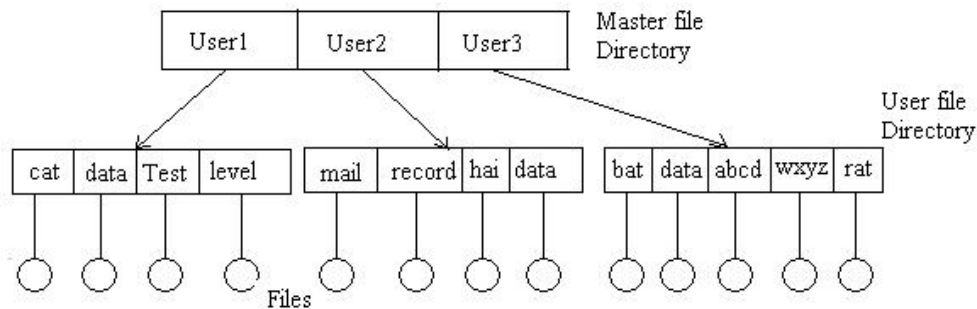
- The simplest directory structure is the single- level directory.
- All files are contained in the same directory.
- **Disadvantage:**
 - When the number of files increases or when the system has more than one user, since all files are in the same directory, they must have unique names.

2. Two – Level Directory

- In the two level directory structures, each user has her own user file directory (UFD).
- When a user job starts or a user logs in, the system's master file directory (MFD) is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user.
- When a user refers to a particular file, only his own UFD is searched.
- Thus, different users may have files with the same name.
- Although the two – level directory structure solves the name-collision problem

Disadvantage:

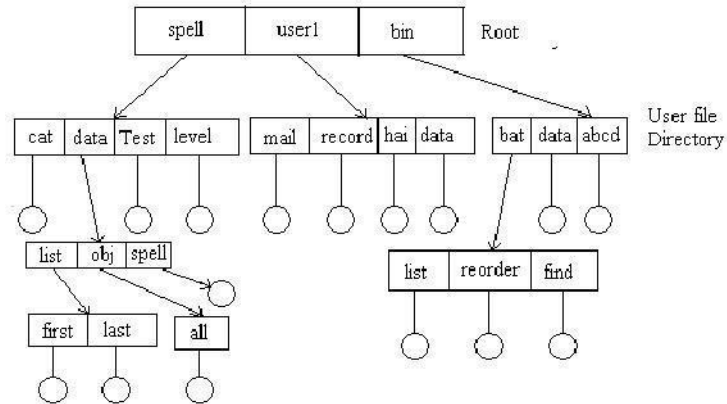
- Users cannot create their own sub-directories.



3. Tree – Structured Directory

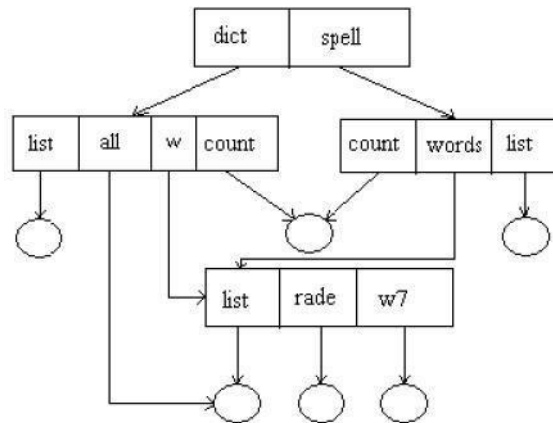
- A tree is the most common directory structure.
- The tree has a root directory. Every file in the system has a unique path name.
- A **path name** is the path from the root, through all the subdirectories to a specified file.
- A directory (or sub directory) contains a set of files or sub directories.
- A directory is simply another file. But it is treated in a special way.
- All directories have the same internal format.

- One bit in each directory entry defines the entry as a file (0) or as a subdirectory (1).
- Special system calls are used to create and delete directories.
- Path names can be of two types: absolute path names or relative path names.
- An absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path.
- A relative path name defines a path from the current directory.



4. Acyclic Graph Directory.

- An acyclic graph is a graph with no cycles.
- To implement shared files and subdirectories this directory structure is used.
- An acyclic – graph directory structure is more flexible than is a simple tree structure, but it is also more complex. In a system where sharing is implemented by symbolic link, this situation is somewhat easier to handle. The deletion of a link does not need to affect the original file; only the link is removed.
- Another approach to deletion is to preserve the file until all references to it are deleted. To implement this approach, we must have some mechanism for determining that the last reference to the file has been deleted.



Sharing and Protection

File Sharing

1. Multiple Users:

- When an operating system accommodates multiple users, the issues of file sharing, file naming and file protection become preeminent.
- The system either can allow user to access the file of other users by default, or it may require that a user specifically grant access to the files.
- These are the issues of access control and protection.
- To implementing sharing and protection, the system must maintain more file and directory attributes than a on a single-user system.
- The owner is the user who may change attributes, grand access, and has the most control over the file or directory.
- The group attribute of a file is used to define a subset of users who may share access to the file.
- Most systems implement owner attributes by managing a list of user names and associated user identifiers (user Ids).
- When a user logs in to the system, the authentication stage determines the appropriate user ID for the user. That user ID is associated with all of user's processes and threads. When they need to be user readable, they are translated, back to the user name via the user name list. Likewise, group functionality can be implemented as a system wide list of group names and group identifiers.
- Every user can be in one or more groups, depending upon operating system design decisions. The user's group Ids is also included in every associated process and thread.

2. Remote File System:

- Networks allowed communications between remote computers.
- Networking allows the sharing or resource spread within a campus or even around the world.
- User manually transfer files between machines via programs like **ftp**.
- A **distributed file system** (DFS) in which remote directories is visible from the local machine.
- The **World Wide Web**: A browser is needed to gain access to the remote file and separate operations (essentially a wrapper for ftp) are used to transfer files.

a) The client-server Model:

- Remote file systems allow a computer to a mount one or more file systems from one or more remote machines.
- A server can serve multiple clients, and a client can use multiple servers, depending on the implementation details of a given client –server facility.
- Client identification is more difficult. Clients can be specified by their network name or other identifier, such as IP address, but these can be spoofed (or imitate). An unauthorized client can spoof the server into deciding that it is authorized, and the unauthorized client could be allowed access.

b) Distributed Information systems:

- Distributed information systems, also known as distributed naming service, have been devised to provide a unified access to the information needed for remote computing.
- Domain name system (DNS) provides host-name-to-network address translations for theirentire Internet (including the World Wide Web).
- Before DNS was invented and became widespread, files containing the same information were sent via e-mail of ftp between all networked hosts.

c) Failure Modes:

- **Redundant arrays of inexpensive disks (RAID)** can prevent the loss of a disk from resulting in the loss of data.

- Remote file system has more failure modes. By nature of the complexity of networking system and the required interactions between remote machines, many more problems can interfere with the proper operation of remote file systems.

d) Consistency Semantics:

- It is characterization of the system that specifies the semantics of multiple users accessing a shared file simultaneously.
- These semantics should specify when modifications of data by one user are observable by other users.
- The semantics are typically implemented as code with the file system.
- A series of file accesses (that is reads and writes) attempted by a user to the same file is always enclosed between the open and close operations.
- The series of access between the open and close operations is a **file session**.

(i) UNIX Semantics:

The UNIX file system uses the following consistency semantics:

1. Writes to an open file by a user are visible immediately to other users that have this file open at the same time.
2. One mode of sharing allows users to share the pointer of current location into the file. Thus, the advancing of the pointer by one user affects all sharing users.

(ii) Session Semantics:

The Andrew file system (AFS) uses the following consistency semantics:

1. Writes to an open file by a user are not visible immediately to other users that have the same file open simultaneously.
2. Once a file is closed, the changes made to it are visible only in sessions starting later. Already open instances of the file do not reflect this change.

(iii) Immutable –shared File Semantics:

- Once a file is declared as shared by its creator, it cannot be modified.
- An immutable file has two key properties:
 - Its name may not be reused and its contents may not be altered.

File Protection

(i) Need for file protection.

- When information is kept in a computer system, we want to keep it safe from **physical damage** (reliability) and **improper access** (protection).
- Reliability is generally provided by duplicate copies of files. Many computers have systems programs that automatically (or through computer-operator intervention) copy disk files to tape at regular intervals (once per day or week or month) to maintain a copy should a file system be accidentally destroyed.
- File systems can be damaged by hardware problems (such as errors in reading or writing), power surges or failures, head crashes, dirt, temperature extremes, and vandalism. Files may be deleted accidentally. Bugs in the file-system software can also cause file contents to be lost.
- Protection can be provided in many ways. For a small single-user system, we might provide protection by physically removing the floppy disks and locking them in a desk drawer or file cabinet. In a multi-user system, however, other mechanisms are needed.

(ii) Types of Access

- Complete protection is provided by prohibiting access.
- Free access is provided with no protection.
- Both approaches are too extreme for general use.

- What is needed is **controlled access**.
- Protection mechanisms provide controlled access by limiting the types of file access that can be made. Access is permitted or denied depending on several factors, one of which is the type of access requested. Several different types of operations may be controlled:

1. **Read:** Read from the file.
2. **Write:** Write or rewrite the file.
3. **Execute:** Load the file into memory and execute it.
4. **Append:** Write new information at the end of the file.
5. **Delete:** Delete the file and free its space for possible reuse.
6. **List:** List the name and attributes of the file.

(iii) Access Control

- Associate with each file and directory an access-control list (ACL) specifying the user name and the types of access allowed for each user.
- When a user requests access to a particular file, the operating system checks the access list associated with that file. If that user is listed for the requested access, the access is allowed. Otherwise, a protection violation occurs and the user job is denied access to the file.
- This technique has two undesirable consequences:
 - Constructing such a list may be tedious and unrewarding task, especially if we do not know in advance the list of users in the system.
 - The directory entry, previously of fixed size, now needs to be of variable size, resulting in more complicated space management.
- To condense the length of the access control list, many systems recognize three classifications of users in connection with each file:
 - **Owner:** The user who created the file is the owner.
 - **Group:** A set of users who are sharing the file and need similar access is a group, or work group.
 - **Universe:** All other users in the system constitute the universe.

File System Implementation- File System Structure

- **Disk** provide the bulk of secondary storage on which a file system is maintained.
- **Characteristics of a disk:**
 1. They can be rewritten in place, it is possible to read a block from the disk, to modify the block and to write it back into the same place.
 2. They can access directly any given block of information to the disk.
 - To produce an efficient and convenient access to the disk, the operating system imposes one or more file system to allow the data to be stored, located and retrieved easily.
 - The file system itself is generally composed of many different levels. Each level in the design uses the features of lower level to create new features for use by higher levels.

Layered File System

- The **I/O control** consists of device drivers and interrupt handlers to transfer information between the main memory and the disk system.
- The **basic file system** needs only to issue generic commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by its numeric disk address (for example, drive -1, cylinder 73, track 2, sector 10)

Directory Implementation

1. Linear List

- The simplest method of implementing a directory is to use a linear list of file names with pointer to the data blocks.
- A linear list of directory entries requires a linear search to find a particular entry.
- This method is simple to program but time-consuming to execute. To create a new file, we must first search the but time – consuming to execute.
- The real disadvantage of a linear list of directory entries is the linear search to find a file.

2. Hash Table

- In this method, a linear list stores the directory entries, but a hash data structure is also used.
- The hash table takes a value computed from the file name and returns a pointer to the file name in the linear list.
- Therefore, it can greatly decrease the directory search time.
- Insertion and deleting are also fairly straight forward, although some provision must be made for collisions – situation where two file names hash to the same location.
- The major difficulties with a hash table are its generally fixed size and the dependence of the hash function on that size.

Allocation Methods

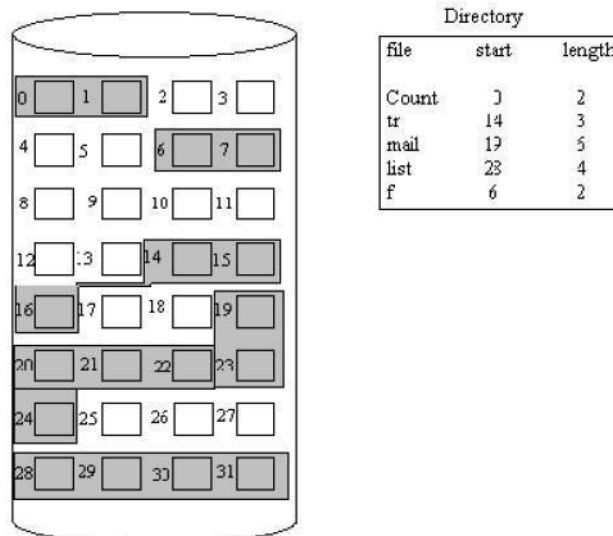
- The main problem is how to allocate space to these files so that disk space is utilized effectively and files can be accessed quickly.

- There are three major methods of allocating disk space:

1. Contiguous Allocation
2. Linked Allocation
3. Indexed Allocation

1. Contiguous Allocation

- The contiguous – allocation method requires each file to occupy a set of contiguous blocks on the disk.



- Contiguous allocation of a file is defined by the disk address and length (in block units) of the first block. If the file is n blocks long and starts at location b , then it occupies blocks $b, b+1, b+2, \dots, b+n-1$.

- The directory entry for each file indicates the address of the starting block and the length of the area allocated for this file.

Disadvantages:

1. Finding space for a new file.

- The contiguous disk space-allocation problem suffer from the problem of external fragmentation. As file are allocated and deleted, the free disk space is broken into chunks. It becomes a problem when the largest contiguous chunk is insufficient for a request; storage is fragmented into a number of holes, no one of which is large enough to store the data.

2. Determining how much space is needed for a file.

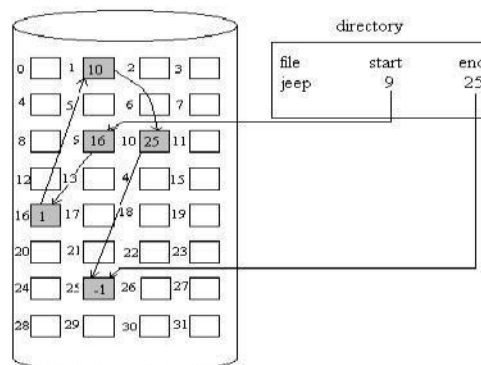
- When the file is created, the total amount of space it will need must be found an allocated how does the creator know the size of the file to be created?
- If we allocate too little space to a file, we may find that file cannot be extended. The other possibility is to find a larger hole, copy the contents of the file to the new space, and release the previous space. This series of actions may be repeated as long as space exists, although it can be time – consuming. However, in this case, the user never needs to be informed explicitly about what is happening ; the system continues despite the problem, although more and more slowly.
- Even if the total amount of space needed for a file is known in advance pre-allocation may be inefficient.
- A file that grows slowly over a long period (months or years) must be allocated enough space for its final size, even though much of that space may be unused for a long time the file, therefore has a large amount of internal fragmentation.

To overcome these disadvantages:

- Use a modified contiguous allocation scheme, in which a contiguous chunk of space called as an **extent** is allocated initially and then, when that amount is not large enough another chunk of contiguous space an extent is added to the initial allocation.
- Internal fragmentation can still be a problem if the extents are too large, and external fragmentation can be a problem as extents of varying sizes are allocated and deallocated.

2. Linked Allocation

- Linked allocation solves all problems of contiguous allocation.
- With linked allocation, each file is a linked list of disk blocks, the disk blocks may be scattered any where on the disk.
- The directory contains a pointer to the first and last blocks of the file. For example, a file of five blocks might start at block 9, continue at block 16, then block 1, block 10, and finally block 25.
- Each block contains a pointer to the next block. These pointers are not made available to the user.
- There is no external fragmentation with linked allocation, and any free block on the free space list can be used to satisfy a request.
- The size of a file does not need to be declared when that file is created. A file can continue to grow as long as free blocks are available consequently, it is never necessary to compact disk space.



Disadvantages:

1. Used effectively only for sequential access files.

- To find the *i*th block of a file, we must start at the beginning of that file, and follow the pointers until we get to the *i*th block. Each access to a pointer requires a disk read, and sometimes a disk seek consequently, it is inefficient to support a direct-access capability for linked allocation files.

2. Space required for the pointers

- If a pointer requires 4 bytes out of a 512-byte block, then 0.78 percent of the disk is being used for pointers, rather than for information.
- Solution to this problem is to collect blocks into multiples, called **clusters**, and to allocate the clusters rather than blocks. For instance, the file system may define a cluster as 4 blocks, and operate on the disk in only cluster units.

3. Reliability

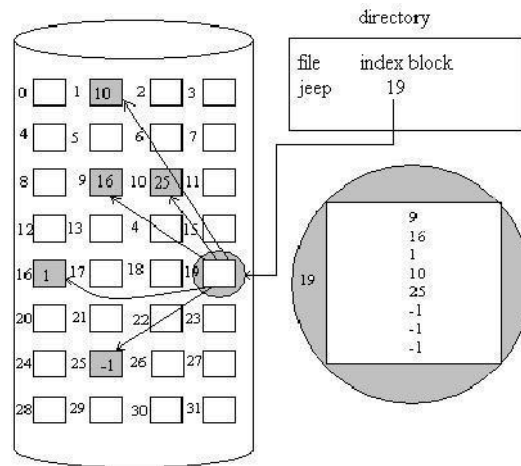
- Since the files are linked together by pointers scattered all over the disk hardware failure might result in picking up the wrong pointer. This error could result in linking into the free-space list or into another file. Partial solutions are to use doubly linked lists or to store the file names in a relative block number in each block; however, these schemes require even more overhead for each file.

File Allocation Table(FAT)

- An important variation on the linked allocation method is the use of a file allocation table(FAT).
- This simple but efficient method of disk-space allocation is used by the MS-DOS and OS/2 operating systems.
- A section of disk at beginning of each partition is set aside to contain the table.
- The table has entry for each disk block, and is indexed by block number.
- The FAT is much as is a linked list.
- The directory entry contains the block number the first block of the file.
- The table entry indexed by that block number contains the block number of the next block in the file.
- This chain continues until the last block which has a special end-of-file value as the table entry.
- Unused blocks are indicated by a 0 table value.
- Allocating a new block file is a simple matter of finding the first 0-valued table entry, and replacing the previous end-of-file value with the address of the new block.
- The 0 is replaced with the end-of-file value, an illustrative example is the FAT structure for a file consisting of disk blocks 217,618, and 339.

3. Indexed Allocation

- Linked allocation solves the external-fragmentation and size-declaration problems of contiguous allocation.
- Linked allocation cannot support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all over the disk and need to be retrieved in order.
- Indexed allocation solves this problem by bringing all the pointers together into one location: the **index block**.
- Each file has its own index block, which is an array of disk-block addresses.
- The *i*th entry in the index block points to the *i*th block of the file.
- The directory contains the address of the index block.
- To read the *i*th block, we use the pointer in the *i*th index-block entry to find and read the desired block this scheme is similar to the paging scheme.



- When the file is created, all pointers in the pointers in the index block are set to nil. when the ith block is first written, a block is obtained from the free space manager, and its address is put in the ith index – block entry.
- Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk may satisfy a request for more space.

Disadvantages

1. Pointer Overhead

- Indexed allocation does suffer from wasted space. The pointer over head of the index block is generally greater than the pointer over head of linked allocation.

2. Size of Index block

If the index block is too small, however, it will not be able to hold enough pointers for a large file, and a mechanism will have to be available to deal with this issue:

- **Linked Scheme:** An index block is normally one disk block. Thus, it can be read and written directly by itself. To allow for large files, we may link together several index blocks.
- **Multilevel index:** A variant of the linked representation is to use a first level index block to point to a set of second – level index blocks.
- **Combined scheme:**
 - o Another alternative, used in the UFS, is to keep the first, say, 15 pointers of the index block in the file's inode.
 - o The first 12 of these pointers point to direct blocks; that is for small (no more than 12 blocks) files do not need a separate index block
 - o The next pointer is the address of a single indirect block.
 - The single indirect block is an index block, containing not data, but rather the addresses of blocks that do contain data.
 - o Then there is a double indirect block pointer, which contains the address of a block that contain pointers to the actual data blocks. The last pointer would contain pointers to the actual data blocks.
 - o The last pointer would contain the address of a triple indirect block.

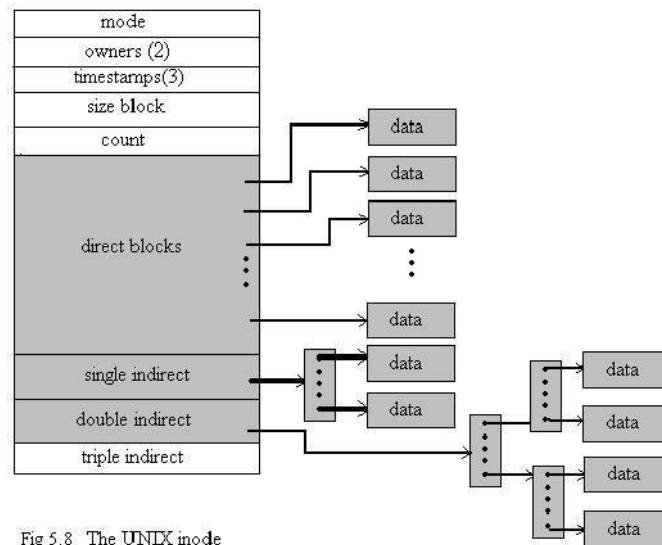


Fig 5.8 The UNIX inode

Free-space Management

- Since disk space is limited, we need to reuse the space from deleted files for new files, if possible.
- To keep track of free disk space, the system maintains a free-space list.
- The free-space list records all free disk blocks – those not allocated to some file or directory.
- To create a file, we search the free-space list for the required amount of space, and allocate that space to the new file.
- This space is then removed from the free-space list.
- When a file is deleted, its disk space is added to the free-space list.

1. Bit Vector

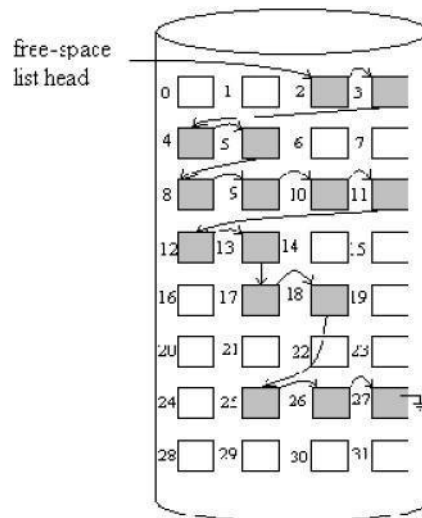
- The free-space list is implemented as a bit map or bit vector.
- Each block is represented by 1 bit. If the block is free, the bit is 1; if the block is allocated, the bit is 0.
- For example, consider a disk where block 2,3,4,5,8,9,10,11,12,13,17,18,25,26 and 27 are free, and the rest of the block are allocated. The free space bit map would be

00111100111111000110000011100000 ...

- The main **advantage** of this approach is its relatively simplicity and efficiency in finding the first free block, or n consecutive free blocks on the disk.

2. Linked List

- Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory.
- This first block contains a pointer to the next free disk block, and so on.
- In our example, we would keep a pointer to block 2, as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on.
- However, this scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time.
- The FAT method incorporates free-block accounting data structure. No separate method is needed.



3. Grouping

- A modification of the free-list approach is to store the addresses of n free blocks in the first free block.
- The first $n-1$ of these blocks are actually free.
- The last block contains the addresses of another n free blocks, and so on.
- The importance of this implementation is that the addresses of a large number of free blocks can be found quickly.

4. Counting

- We can keep the address of the first free block and the number n of free contiguous blocks that follow the first block.
- Each entry in the free-space list then consists of a disk address and a count.
- Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than 1.

I/O Systems

I/O Hardware

The role of the operating system in computer I/O is to manage and control I/O operations and I/O devices. A device communicates with a computer system by sending signals over a cable or even through the air.

Port: The device communicates with the machine via a connection point (or port), for example, a serial port.

Bus: If one or more devices use a common set of wires, the connection is called a bus.

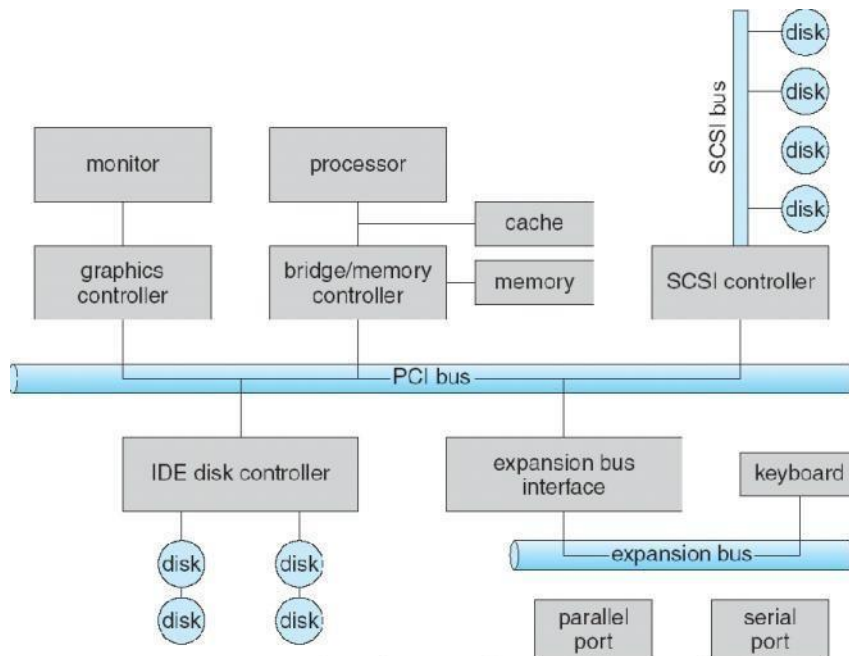
Daisy chain: Device A' has a cable that plugs into device B', and device B' has a cable that plugs into device C', and device C' plugs into a port on the computer, this arrangement is called a daisy chain. A daisy chain usually operates as a bus.

PC bus structure:

A PCI bus that connects the processor-memory subsystem to the fast devices, and an expansion bus that connects relatively slow devices such as the keyboard and serial and parallel ports. In the upper-right portion of the figure, four disks are connected together on a SCSI bus plugged into a SCSI controller.

A **controller or host adapter** is a collection of electronics that can operate a port, a bus, or a device. A serial-port controller is a simple device controller. It is a single chip in the computer that controls the signals on the wires of a serial port. By contrast, a SCSI bus controller is not simple. Because the SCSI

protocol is complex, the SCSI bus controller is often implemented as a separate circuit board. It typically contains a processor, microcode, and some private memory. Some devices have their own built-in controllers.



• How can the processor give commands and data to a controller to accomplish an I/O transfer?

- Direct I/O instructions
- Memory-mapped I/O

Direct I/O instructions

Use special I/O instructions that specify the transfer of a byte or word to an I/O port address. The I/O instruction triggers bus lines to select the proper device and to move bits into or out of a device register

Memory-mapped I/O

The device-control registers are mapped into the address space of the processor. The CPU executes I/O requests using the standard data-transfer instructions to read and write the device-control registers.

An I/O port typically consists of four registers: status, control, data-in, and data-out registers.

Status register	Read by the host to indicate states such as whether the current command has completed, whether a byte is available to be read from the data-in register, and whether there has been a device error.
Control register	Written by the host to start a command or to change the mode of a device.
data-in register	Read by the host to get input
data-out register	Written by the host to send output

Polling

Interaction between the host and a controller

- The controller sets the busy bit when it is busy working, and clears the busy bit when it is ready to accept the next command.
- The host sets the command ready bit when a command is available for the controller to execute.

Coordination between the host & the controller is done by handshaking as follows:

1. The host repeatedly reads the busy bit until that bit becomes clear.
2. The host sets the write bit in the command register and writes a byte into the data-out register.
3. The host sets the command-ready bit.
4. When the controller notices that the command-ready bit is set, it sets the busy bit.
5. The controller reads the command register and sees the write command. It reads the data-out register to get the byte, and does the I/O to the device.
6. The controller clears the command-ready bit, clears the error bit in the status register to indicate that the device I/O succeeded, and clears the busy bit to indicate that it is finished.

In step 1, the host is **busy-waiting or polling** : It is in a loop, reading the status register over and over until the busy bit becomes clear.

Interrupts

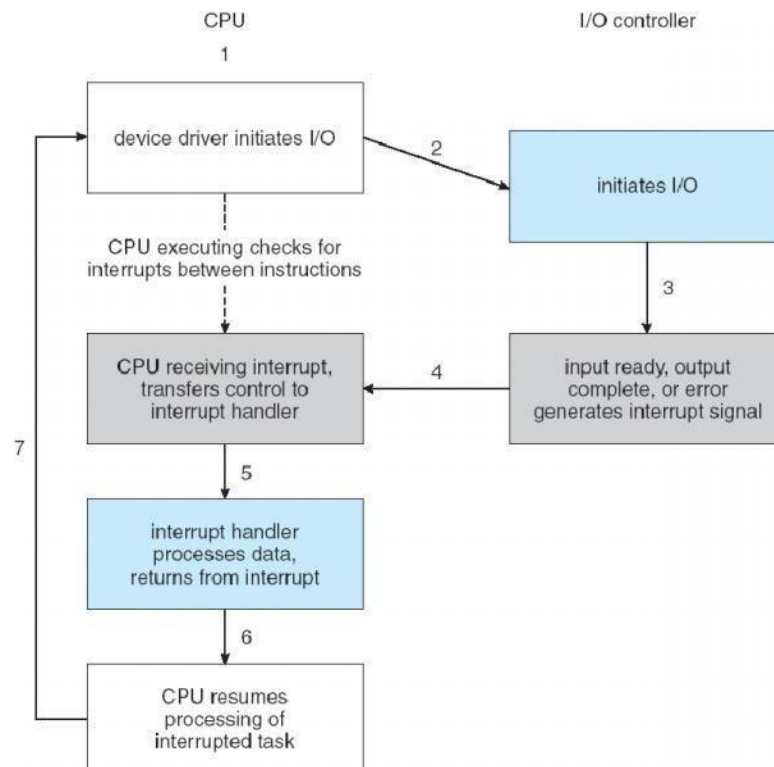
The CPU hardware has a wire called the interrupt-request line .

The basic interrupt mechanism works as follows;

1. Device controller raises an interrupt by asserting a signal on the interrupt request line.
2. The CPU catches the interrupt and dispatches to the interrupt handler and
3. The handler clears the interrupt by servicing the device.

Two interrupt request lines:

1. **Nonmaskable interrupt**: which is reserved for events such as unrecoverable memory errors?
2. **Maskable interrupt**: Used by device controllers to request service



Application I/O Interface

- I/O system calls encapsulate device behaviors in generic classes
- Device-driver layer hides differences among I/O controllers from kernel

- Devices vary in many dimensions
 1. Character-stream or block
 2. Sequential or random-access
 3. Sharable or dedicated
 4. Speed of operation
 5. read-write, read only, or write only

Types	Description	Example
Character-stream or block	A character-stream device transfers bytes one by one, whereas a block device transfers a block of bytes as a unit.	Terminal, Disk
Sequential or random-access	A sequential device transfers data in a fixed order determined by the device, whereas the user of a random-access device can instruct the device to seek to any of the available data storage locations.	Modem, CD-ROM
Sharable or dedicated	A sharable device can be used concurrently by several processes or threads; a dedicated device cannot.	Tape, Keyboard
Speed of operation	Latency, seek time, transfer rate, delay between operations	
read-write, read only, or write only	Some devices perform both input and output, but others support only one data direction.	CD-ROM, Graphics controller, Disk

Block and Character Devices

Block-device: The block-device interface captures all the aspects necessary for accessing disk drives and other block-oriented devices. The device should understand the commands such as read () & write (), and if it is a random access device, it has a seek() command to specify which block to transfer next.

Applications normally access such a device through a file-system interface. The OS itself, and special applications such as database-management systems, may prefer to access a block device as a simple linear array of blocks. This mode of access is sometimes called **raw I/O**.

Memory-mapped file access can be layered on top of block-device drivers. Rather than offering read and write operations, a memory-mapped interface provides access to disk storage via an array of bytes in main memory.

Character Devices: A keyboard is an example of a device that is accessed through a character stream interface. The basic system calls in this interface enable an application to get() or put() one character.

On top of this interface, libraries can be built that offer line-at-a-time access, with buffering and editing services.

(+) This style of access is convenient for input devices where it produce input "spontaneously".

(+) This access style is also good for output devices such as printers or audio boards, which naturally fit the concept of a linear stream of bytes.

Network Devices

Because the performance and addressing characteristics of network I/O differ significantly from those of disk I/O, most operating systems provide a network I/O interface that is different from the read0 -write() - seek() interface used for disks.

- Windows NT provides one interface to the network interface card, and a second interface to the network protocols.
- In UNIX, we find half-duplex pipes, full-duplex FIFOs, full-duplex STREAMS, message queues and sockets.

Clocks and Timers

Most computers have hardware clocks and timers that provide three basic functions:

1. Give the current time
2. Give the elapsed time
3. Set a timer to trigger operation X at time T

These functions are used by the operating system & also by time sensitive applications. Programmable interval timer: The hardware to measure elapsed time and to trigger operations is called a programmable interval timer. It can be set to wait a certain amount of time and then to generate an interrupt. To generate periodic interrupts, it can be set to do this operation once or to repeat.

Blocking and Non-blocking I/O (or) synchronous & asynchronous:

Blocking I/O: When an application issues a blocking system call;

- The execution of the application is suspended.
- The application is moved from the operating system's run queue to a wait queue.
 - After the system call completes, the application is moved back to the run queue, where it is eligible to resume execution, at which time it will receive the values returned by the system call.

Non-blocking I/O: Some user-level processes need non-blocking I/O.

Examples: 1. User interface that receives keyboard and mouse input while processing and displaying data on the screen.

2. Video application that reads frames from a file on disk while simultaneously decompressing and displaying the output on the display.

Kernel I/O Subsystem

Kernels provide many services related to I/O.

- One way that the I/O subsystem improves the efficiency of the computer is by scheduling I/O operations.
 - Another way is by using storage space in main memory or on disk, via techniques called buffering, caching, and spooling.

I/O Scheduling:

To determine a good order in which to execute the set of I/O requests.

Uses:

- a) It can improve overall system performance,
- b) It can share device access fairly among processes, and
- c) It can reduce the average waiting time for I/O to complete.

Implementation: OS developers implement scheduling by maintaining a queue of requests for each device.

1. When an application issues a blocking I/O system call,
2. The request is placed on the queue for that device.

3. The I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by applications.

Buffering:

Buffer: A memory area that stores data while they are transferred between two devices or between a device and an application.

Reasons for buffering:

- a) To cope with a speed mismatch between the producer and consumer of a data stream.
- b) To adapt between devices that have different data-transfer sizes.
- c) To support copy semantics for application I/O.

Copy semantics: Suppose that an application has a buffer of data that it wishes to write to disk. It calls the write () system call, providing a pointer to the buffer and an integer specifying the number of bytes to write.

After the system call returns, what happens if the application changes the contents of the buffer? With copy semantics, the version of the data written to disk is guaranteed to be the version at the time of the application system call, independent of any subsequent changes in the application's buffer.

A simple way that the operating system can guarantee copy semantics is for the write() system call to copy the application data into a kernel buffer before returning control to the application. The disk write is performed from the kernel buffer, so that subsequent changes to the application buffer have no effect.

Caching

A cache is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original

Cache vs buffer: A buffer may hold the only existing copy of a data item, whereas a cache just holds a copy on faster storage of an item that resides elsewhere.

When the kernel receives a file I/O request,

1. The kernel first accesses the buffer cache to see whether that region of the file is already available in main memory.
2. If so, a physical disk I/O can be avoided or deferred. Also, disk writes are accumulated in the buffer cache for several seconds, so that large transfers are gathered to allow efficient write schedules.

Spooling and Device Reservation:

Spool: A buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. A printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together

The os provides a control interface that enables users and system administrators ;

- a) To display the queue,
- b) To remove unwanted jobs before those jobs print,
- c) To suspend printing while the printer is serviced, and so on.

Device reservation - provides exclusive access to a device

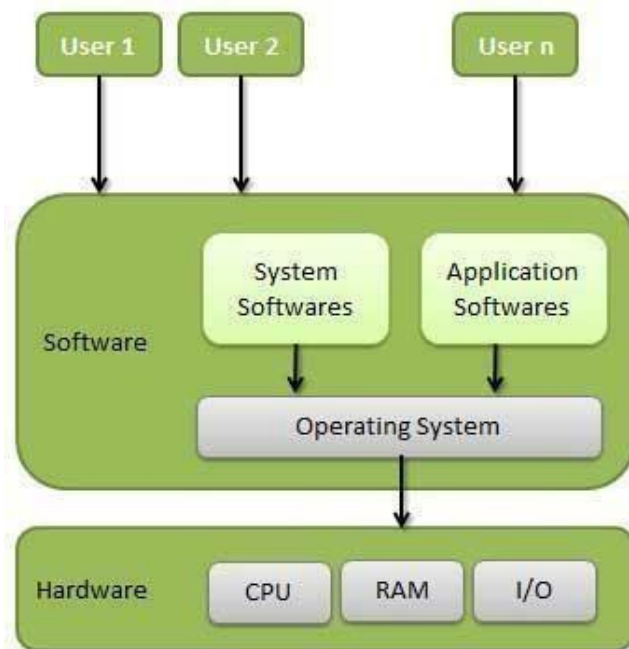
- System calls for allocation and de-allocation
- Watch out for deadlock

Error Handling:

- An operating system that uses protected memory can guard against many kinds of hardware and application errors.
- OS can recover from disk read, device unavailable, transient write failures
- Most return an error number or code when I/O request fails
- System error logs hold problem reports.

5.1 The Linux System

- An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs. The Linux open source operating system, or Linux OS, is a freely distributable, cross-platform operating system based on UNIX.
- The Linux consist of a kernel and some system programs. There are also some application programs for doing work. The kernel is the heart of the operating system which provides a set of tools that are used by system calls.
- The defining component of Linux is the Linux kernel, an operating system kernel first released on 5 October 1991 by *Linus Torvalds*.



- A Linux-based system is a modular Unix-like operating system. It derives much of its basic design from principles established in UNIX. Such a system uses a monolithic kernel which handles process control, networking, and peripheral and file system access.

5.2 Important features of Linux Operating System

- **Portable** - Portability means software can work on different types of hardware in same way. Linux kernel and application programs supports their installation on any kind of hardware platform.
- **Open Source** - Linux source code is freely available and it is community based development project.
- **Multi-User & Multiprogramming** - Linux is a multiuser system where multiple users can access system resources like memory/ ram/ application programs at same time. Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System** - Linux provides a standard file structure in which system files/ user filesar

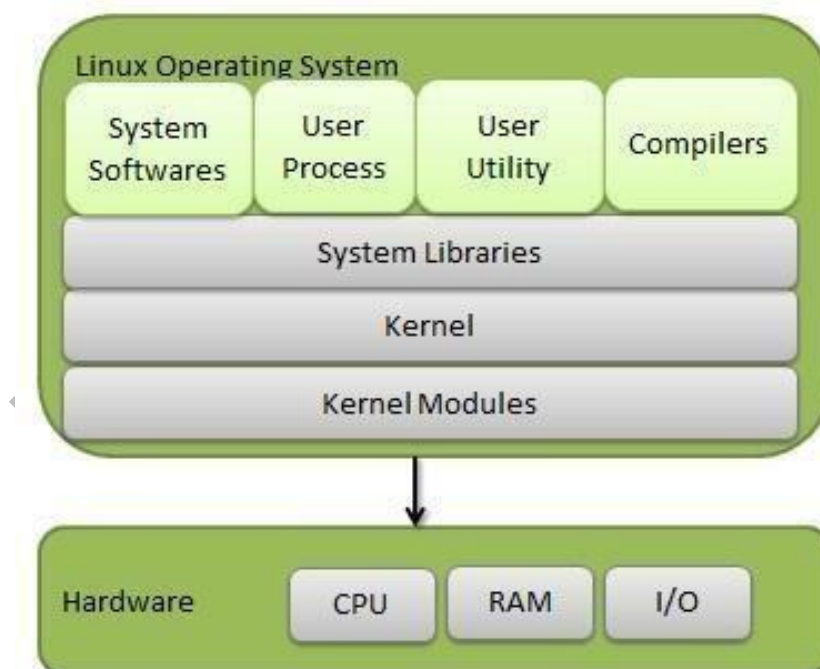
arranged.

- **Shell** - Linux provides a special interpreter program which can be used to execute commands of the operating system.
- **Security** - Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

5.3 Components of Linux System

Linux Operating System has primarily three components

- **Kernel** - Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
- **System Library** - System libraries are special functions or programs using which application programs or system utilities access Kernel's features. These libraries implement most of the functionalities of the operating system and do not require kernel module's code access rights.
- **System Utility** - System Utility programs are responsible to do specialized, individual level tasks

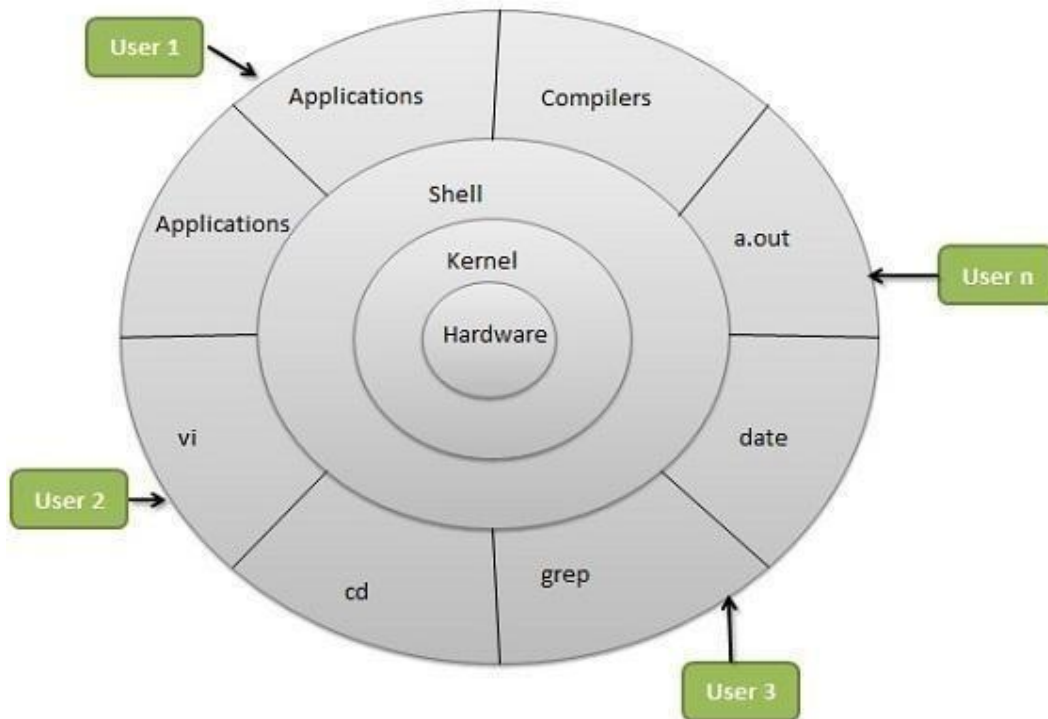


Installed components of a Linux system include the following:

- A **bootloader** is a program that loads the Linux kernel into the computer's main memory, by being executed by the computer when it is turned on and after the firmware initialization is performed.
- An **init** program is the first process launched by the Linux kernel, and is at the root of the process tree.
- **Software libraries**, which contain code that can be used by running processes. The most commonly used software library on Linux systems, the GNU C Library (glibc), C standard library and Widget toolkits.
- **User interface programs** such as command shells or windowing environments. The user

interface, also known as the shell, is either a command-line interface (CLI), a graphical user interface (GUI), or through controls attached

5.4 Architecture



Linux System Architecture is consists of following layers

1. **Hardware layer** - Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
2. **Kernel** - Core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.
3. **Shell** - An interface to kernel, hiding complexity of kernel's functions from users. Takes commands from user and executes kernel's functions.
4. **Utilities** - Utility programs giving user most of the functionalities of an operating systems.

5.5 Modes of operation

- **Kernel Mode:**
 - Kernel component code executes in a special privileged mode called *kernel mode* with full access to all resources of the computer.
 - This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast.
 - Kernel runs each processes and provides system services to processes, provides protected access to hardware to processes.
- **User Mode:**
 - The system programs use the tools provided by the kernel to implement the various services required from an operating system. System programs, and all other programs, run `on top of the kernel', in what is called the user mode.
 - Support code which is not required to run in kernel mode is in System Library.

- User programs and other system programs work in User Mode which has no access to system hardware and kernel code.
- User programs/ utilities use System libraries to access Kernel functions to get system's low level tasks.

5.6 Major Services provided by LINUX System

1. Initialization (**init**)

The single most important service in a LINUX system is provided by **init** program. The **init** is started as the first process of every LINUX system, as the last thing the kernel does when it boots. When **init** starts, it continues the boot process by doing various startup chores (checking and mounting file systems, starting daemons, etc).

2. Logins from terminals (**getty**)

Logins from terminals (via serial lines) and the console are provided by the **getty** program. **init** starts a separate instance of **getty** for each terminal upon which logins are to be allowed. **Getty** reads the username and runs the login program, which reads the password. If the username and password are correct, login runs the shell.

3. Logging and Auditing (**syslog**)

The kernel and many system programs produce error, warning, and other messages. It is often important that these messages can be viewed later, so they should be written to a file. The program doing this logging operation is known as **syslog**.

4. Periodic command execution (**cron & at**)

Both users and system administrators often need to run commands periodically. For example, the system administrator might want to run a command to clean the directories with temporary files from old files, to keep the disks from filling up, since not all programs clean up after themselves correctly.

- The **cron** service is set up to do this. Each user can have a *crontab* file, where the lists the commands wish to execute and the times they should be executed.
- The **at** service is similar to **cron**, but it is once only: the command is executed at the given time, but it is not repeated.

5. Graphical user interface

- UNIX and Linux don't incorporate the user interface into the kernel; instead, they let it be implemented by user level programs. This applies for both text mode and graphical environments. This arrangement makes the system more

flexible.

- The graphical environment primarily used with Linux is called the X Window System (X for short) that provides tools with which a GUI can be implemented. Some popular window managers are blackbox and windowmaker. There are also two popular desktop managers, KDE and Gnome.

6. Network logins (telnet, rlogin & ssh)

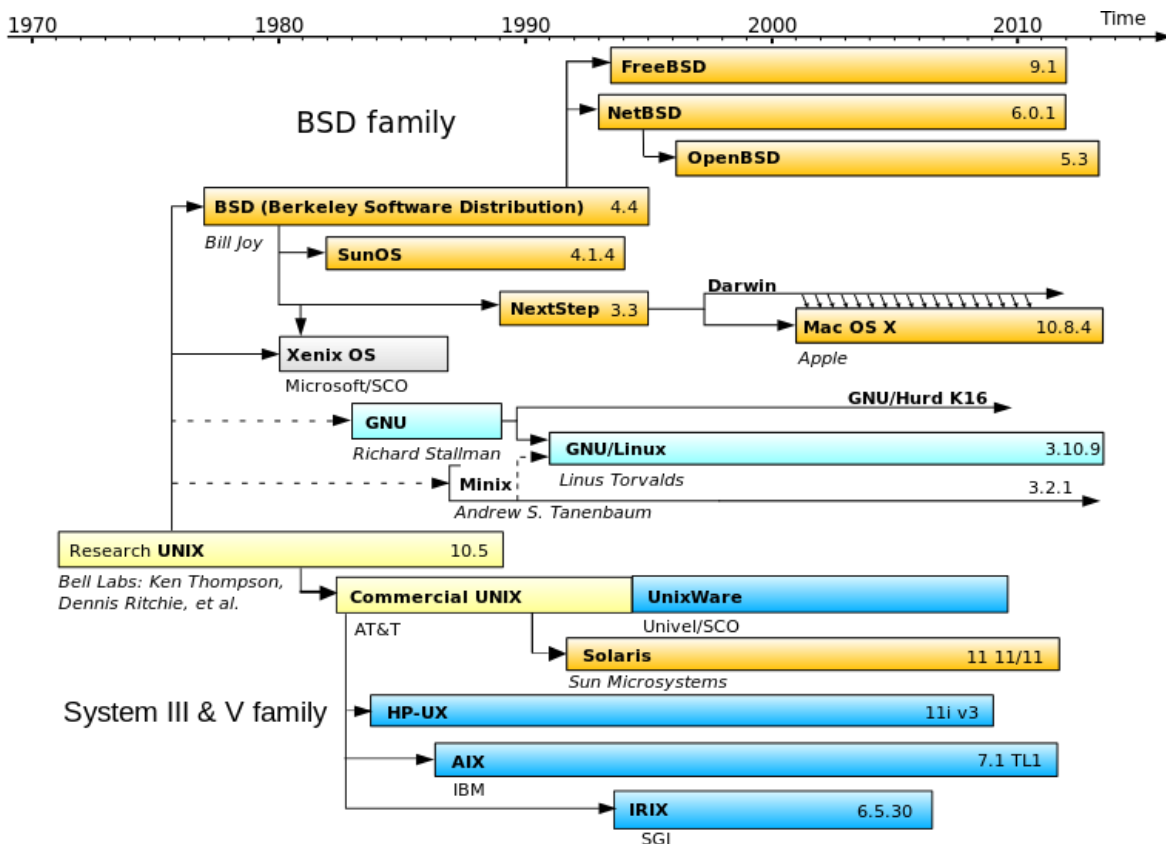
Network logins work a little differently than normal logins. For each person logging in via the network there is a separate virtual network connection. It is therefore not possible to run a separate getty for each virtual connection. There are several different ways to log in via a network, **telnet** and **ssh** being the major ones in TCP/IP networks.

Most of Linux system administrators consider telnet and rlogin to be insecure and prefer ssh, the "secure shell", which encrypts traffic going over the network, thereby making it far less likely that the malicious can "sniff" the connection and gain sensitive data like usernames and passwords.

7. Network File System (NFS & CIFS)

One of the more useful things that can be done with networking services is sharing files via a network file system. Depending on your network this could be done over the Network File System (NFS), or over the Common Internet File System (CIFS).

NFS is typically a 'UNIX' based service. In Linux, NFS is supported by the kernel. CIFS however is not. In Linux, CIFS is supported by **Samba**. With a network file system any file operations done by a program on one machine are sent over the network to another computer.



5.7 SYSTEM ADMINISTRATOR

- A system administrator is a person who is responsible for the configuration and reliable operation of computer systems, especially multi-user computers, such as servers.
- The system administrator seeks to ensure that the uptime, performance, resources, and security of the computers without exceeding the budget.
- To meet these needs, a system administrator may acquire, install, or upgrade computer components and software, provide routine automation, maintain security policies AND troubleshoot.

5.7.1 Responsibilities of a System Administrator

A system administrator's responsibilities might include:

- Installing and configuring new hardware and software.
- Applying operating system updates, patches, and configuration changes.
- Analyzing system logs and identifying potential issues with computer systems.
- Introducing and integrating new technologies into existing data center environments and configuring, adding, and deleting file systems.
- Performing routine audit of systems and software.
- Adding, removing, or updating user account information, resetting passwords, etc.
- Responsibility for security and documenting the configuration of the system.
- Troubleshooting any reported problems.
- System performance tuning.

5.7.2 Various System Administrator Roles

In a larger company, these may all be separate positions within a computer support or Information Services (IS) department. In a smaller group they may be shared by a few sysadmins, or even a single person.

- A **database administrator** (DBA) maintains a database system, and is responsible for the integrity of the data and the efficiency and performance of the system.
- A **network administrator** maintains network infrastructure such as switches and routers, and diagnoses problems with these or with the behaviour of network-attached computers.
- A **security administrator** is a specialist in computer and network security, including the administration of security devices such as firewalls, as well as consulting on general security measures.

- A **web administrator** maintains web server services (such as Apache or IIS) that allow for internal or external access to web sites. Tasks include managing multiple sites, administering security, and configuring necessary components and software.
- A **computer operator** performs routine maintenance and upkeep, such as changing backup tapes or replacing failed drives in a redundant array of independent disks (RAID).
- A **postmaster** administers a mail server.
- A **Storage Administrator (SAN)** can create, provision, add or remove Storage to/from Computer systems. Storage can be attached locally to the system or from a storage area network (SAN) or network-attached storage (NAS).

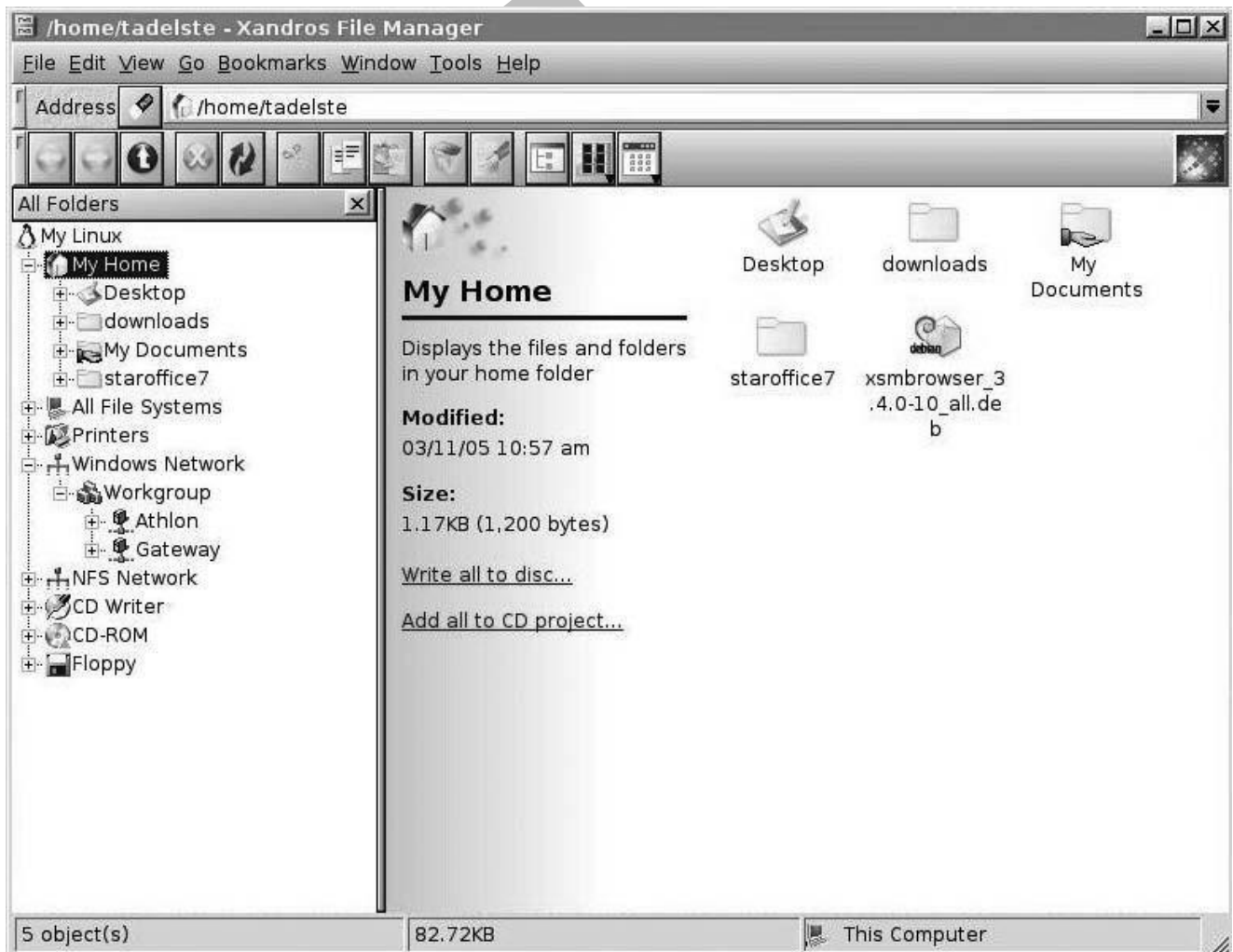
5.7.3 Requirements for LINUX system administrator

1. While specific knowledge is a boon, system administrator should possess basic knowledge about all aspects of Linux. For example, a little knowledge about Solaris, BSD, nginx or various flavors of Linux.
2. Knowledge in at least one of the upper tier scripting language such as Python, Perl, Ruby or more.
3. To be a system administrator, he/she at least needs to have some hands-on experience of system management, system setup and managing Linux or Solaris based servers as well as configuring them.
4. Knowledge in shell programming such as Buorne or Korn and architecture.
5. Knowledge about storage technologies like FC, NFS or iSCSI is great, while knowledge regarding backup technologies is a must for a system administrator.
6. Knowledge in testing methodologies like Subversion or Git is great, while knowledge of version control is also an advantage.
7. Knowledge about basics of configuration management tools like Puppet and Chef.
8. Skills with system and application monitoring tools like SNMP or Nagios are also important, as they show your ability as an administrator in a team setting.
9. Knowledge about how to operate virtualized VMWare or Xen Server, Multifunction Server and Samba
10. An ITIL Foundation certification for Linux system administrator.

5.8 SETTING UP A LINUX MULTIFUNCTION SERVER

A Linux machine can be configured as a server either by compiling several well-defined scripts and off-line downloaded packages or through on-line installation method. Setting up a multifunction server, the system administrator should have knowledge about a series of shell commands. A Linux machine can be configured as any of following application servers such as,

- A Web Server (Apache 2.0.x)
- A Mail Server (Postfix)
- A DNS Server (BIND 9)
- An FTP Server (ProFTPD)
- Mail Delivery Agents (POP3/POP3s/IMAP/IMAPs)
- Webalizer for web site statistics



Files and directories shared by Linux system, as viewed from a
Windows PC

5.8.1 Server Requirements

To set up a Linux Internet server, we will need a connection to the Internet and a static IP address. The system can also be setup with the address leased by ISP and configure it statically.

Computer with at least a Pentium III CPU, a minimum of 256 MB of RAM, and a 10 GB hard drive is preferred. Obviously, a newer CPU and additional memory will provide better performance. This chapter is based on Debian's stable version. We strongly suggest using a CD with the Netinstall kernel. The Debian web site provides downloadable CD images.

5.8.2 Installing & Configuring Network Services

Administrator should log into the server from a remote console on desktop. It is recommended to do further administration from another system (even a laptop), because a secure server normally runs in what is called headless mode—that is, it has no monitor or keyboard.

Get used to administering the server like this. A SSH client on the remote machine is needed which virtually all Linux distributions have and which can be downloaded for other operating systems as well.

Configuring the Network

If DHCP is used during the Debian installation, Server with a static IP address should be configured as follows,

1. To change the settings to use a static IP address, you'll need to become root and edit the file `/etc/network/interfaces` to suit your needs. As an example, we'll use the IP address 70.153.258.42.
2. To add the IP address 70.153.258.42 to the interface eth0, we must change the file to look like this (you'll have to obtain some of the information from your ISP):

```
auto        eth0
iface      eth0 inet static
address    70.153.258.42
netmask    255.255.255.248
network    70.153.258.0
broadcast  70.153.258.47
gateway    70.153.258.46
```

3. After editing the `/etc/network/interfaces` file, restart the network by entering:

```
# /etc/init.d/networking restart
```

4. To edit `/etc/resolv.conf` and add nameservers to resolve Internet hostnames to their corresponding IP addresses. At this point, we will simply set up a minimal DNS

server. Our ***resolv.conf*** looks as follows:

```
search      server
nameserver  70.153.258.42
nameserver  70.253.158.45
nameserver  151.164.1.8
```

5. Now edit `/etc/hosts` and add your IP addresses:

```
127.0.0.1 localhost.localdomain localhost server1
70.153.258.42  server1.centralsoft.org  server1
```

6. Now, to set the hostname, enter these commands:

```
# echo server1.centralsoft.org > /etc/hostname
# /bin/hostname -F /etc/hostname
```

7. verify that you configured your hostname correctly by running the `hostname` command:

```
~$ hostname -f
server1.centralsoft.org
```

5.9 Providing Domain Name Services (BIND - the ubiquitous DNS server)

- Debian provides a stable version of BIND in its repositories. BIND can be installed, setup and secure it in a `chroot` environment, meaning it won't be able to see or access files outside its own directory tree. This is an important security technique.
- The term `chroot` refers to the trick of changing the root filesystem (the `/`directory) that a process sees, so that most of the system is effectively inaccessible to it.
- The BIND server also can be configured to run as a non-root user. That way, if someone gains access to BIND, he/she won't gain root privileges or be able to control other processes.

1. To install BIND on your Debian server, run this command:

```
# apt-get install bind9
```

Debian downloads and configures the file as an Internet service and the status can be seen on the console:

```
Setting up bind9 (9.2.4-1)
Adding group `bind' (104) - Done.
Adding system user `bind'
Adding new user `bind' (104) with group `bind'.
Not creating home directory.
Starting domain name service: named.
```

2. To put BIND in a secured environment, create a directory where the service can run unexposed to other processes. First stop the service by running the following command:

```
# /etc/init.d/bind9 stop
```

3. Edit the file `/etc/default/bind9` so that the daemon will run as the unprivileged user `bind`, chrooted to `/var/lib/named`. Change the line:

```
OPTS="-u bind"
```

So that it reads:

```
OPTIONS="-u bind -t /var/lib/named"
```

4. To provide a complete environment for running BIND, create the necessary directories under `/var/lib`:

```
# mkdir -p /var/lib/named/etc
# mkdir /var/lib/named/dev
# mkdir -p /var/lib/named/var/cache/bind
# mkdir -p /var/lib/named/var/run/bind/run
```

Then move the config directory from `/etc` to `/var/lib/named/etc`:

```
# mv /etc/bind /var/lib/named/etc
```

Next, create a symbolic link to the new config directory from the old location, to avoid problems when BIND is upgraded in the future:

```
# ln -s /var/lib/named/etc/bind /etc/bind
```

Make null and random devices for use by BIND, and fix the permissions of the directories:

```
# mknod /var/lib/named/dev/null c 1 3
# mknod /var/lib/named/dev/random c 1 8
```

Then change permissions and ownership on the files:

```
# chmod 666 /var/lib/named/dev/null
                /var/lib/named/dev/random
# chown -R bind:bind /var/lib/named/var/*
# chown -R bind:bind /var/lib/named/etc/bind
```

5. Finally, start BIND:

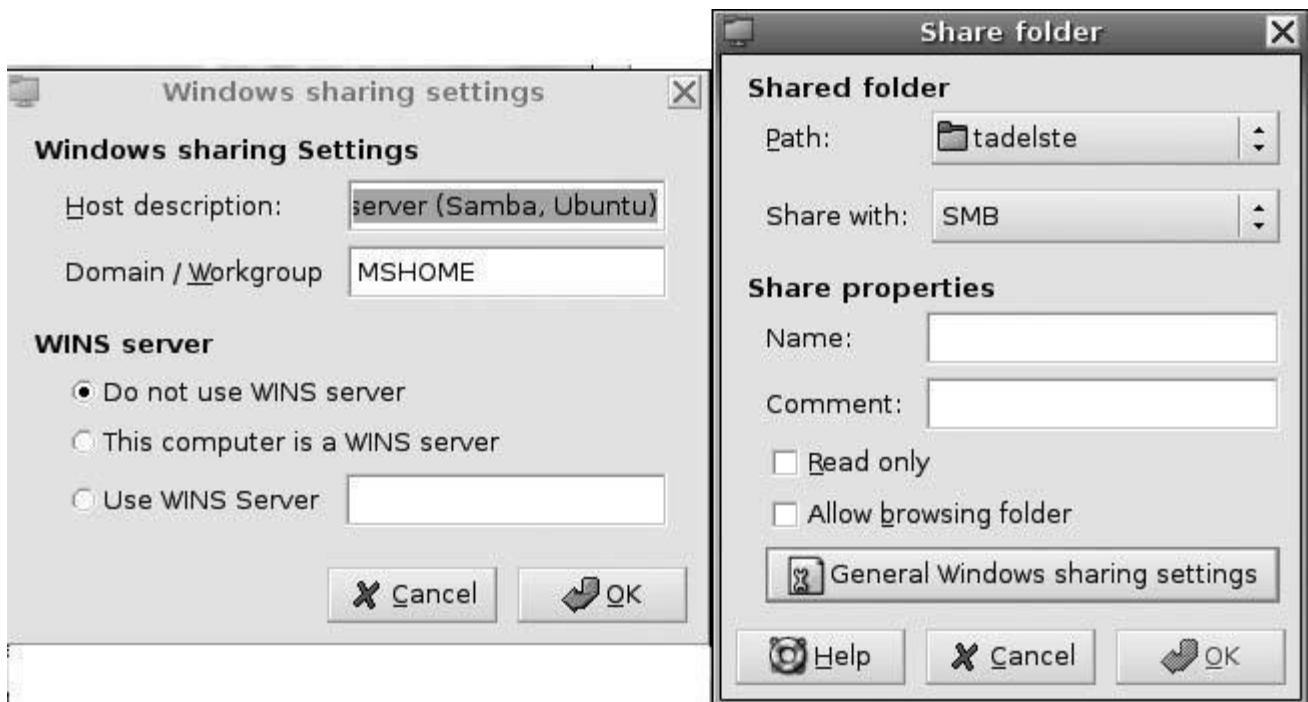
```
# /etc/init.d/bind9 start
```

6. To check whether `named` is functioning without any trouble. Execute this command:

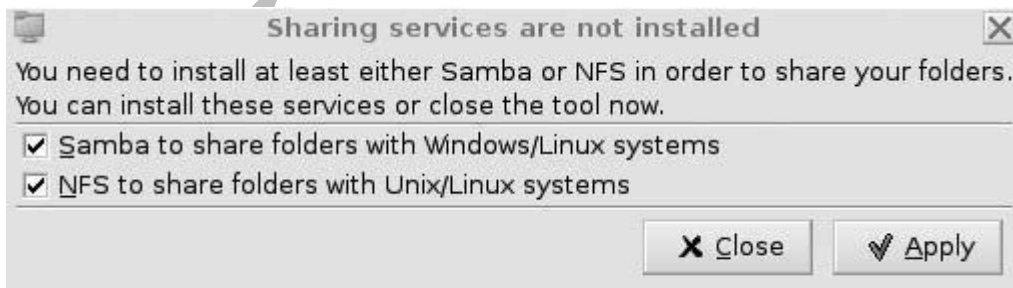
```
server1:/home/admin# rndc status
```

```
number of zones: 6
debug level: 0
xfers running: 0
xfers deferred: 0
soa queries in progress: 0
query logging is OFF
```

```
server is up and running
server1:/home/admin#
```



Setting up Ubuntu shares in a Windows environment



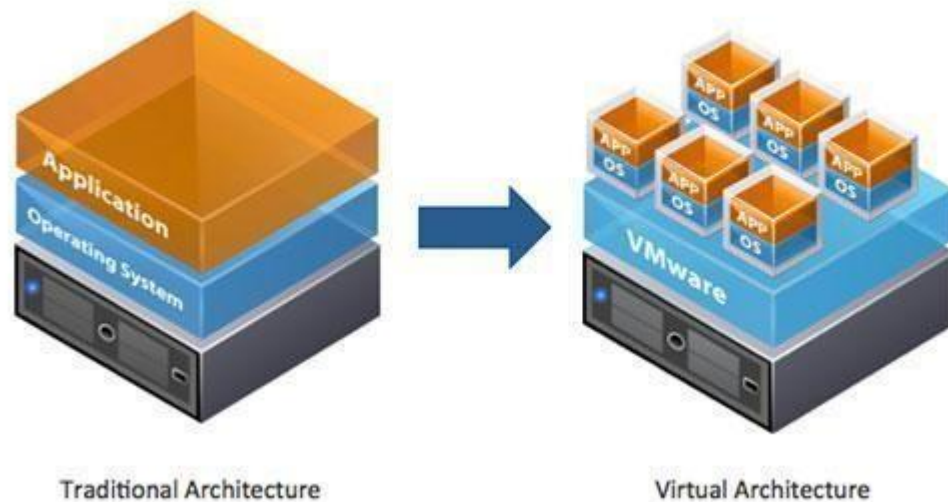
Ubuntu's setup screen for file-sharing

5.10 Virtualization

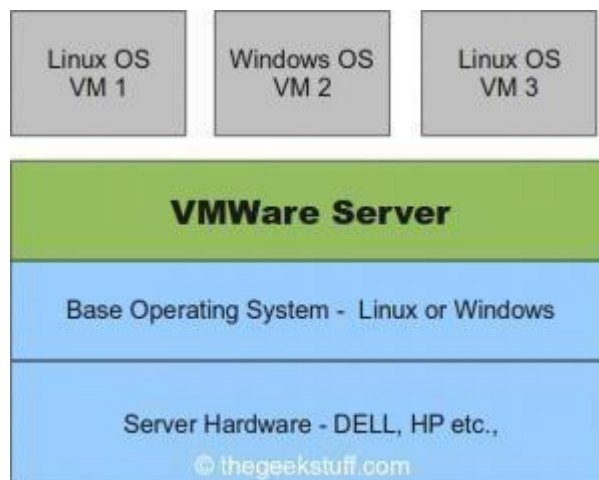
- Virtualization refers to the act of creating a virtual (rather than actual) version of something, including a virtual computer hardware platform, operating system (OS), storage device, or computer network resources.

Virtualization Defined

For those more visually inclined...



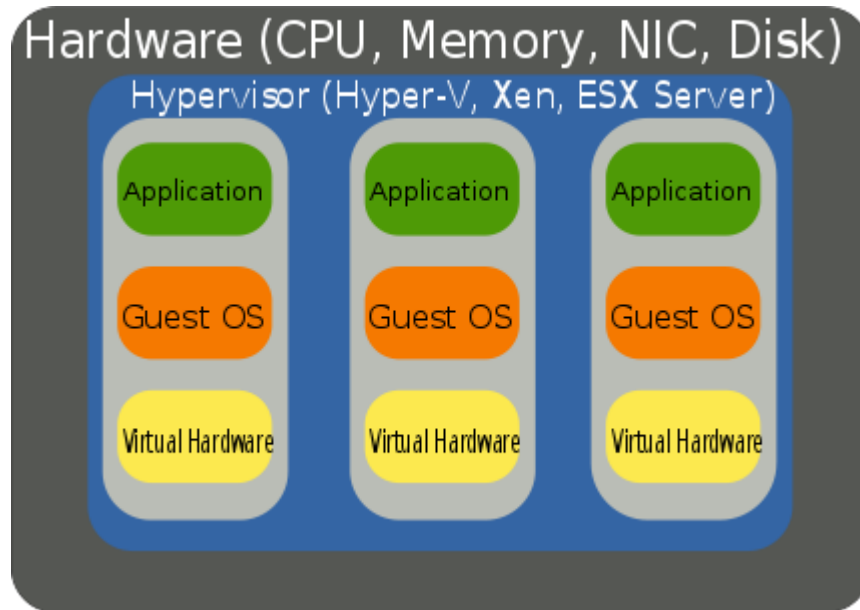
Traditional Architecture vs. Virtual Architecture



Virtual Machine Server – A Layered Approach

- *Hardware virtualization* or *platform virtualization* refers to the creation of a virtual machine that acts like a real computer with an operating system. Software executed on these virtual machines is separated from the underlying hardware resources.
- *Hardware virtualization* hides the physical characteristics of a computing platform from **material**

- For example, a computer that is running Microsoft Windows may host a virtual machine that looks like a computer with the Ubuntu Linux operating system; Ubuntu-based software can be run on the virtual machine.



Hardware Virtualization

Benefits of Virtualization

1. Instead of deploying several physical servers for each service, only one server can be used. Virtualization let multiple OSs and applications to run on a server at a time. Consolidate hardware to get vastly higher productivity from fewer servers.
2. If the preferred operating system is deployed as an image, so we needed to go through the installation process only once for the entire infrastructure.
3. **Improve business continuity:** Virtual operating system images allow us for instant recovery in case of a system failure. The crashed system can be restored back by coping the virtual image.
4. **Increased uptime:** Most server virtualization platforms offer a number of advanced features that just aren't found on physical servers which increases servers' uptime. Some of features are live migration, storage migration, fault tolerance, high availability, and distributed resource scheduling.
5. **Reduce capital and operating costs:** Server consolidation can be done by running multiple virtual machines (VM) on a single physical server. Fewer servers means lower capital and operating costs.

Architecture - Virtualization

The heart of virtualization is the “virtual machine” (VM), a tightly isolated software container with an operating system and application inside. Because each virtual machine is completely separate and independent, many of them can run simultaneously on a single computer. A thin layer of software called a hypervisor decouples the virtual machines from the host and dynamically allocates computing resources to each virtual machine as needed.

This architecture redefines your computing equation and delivers:

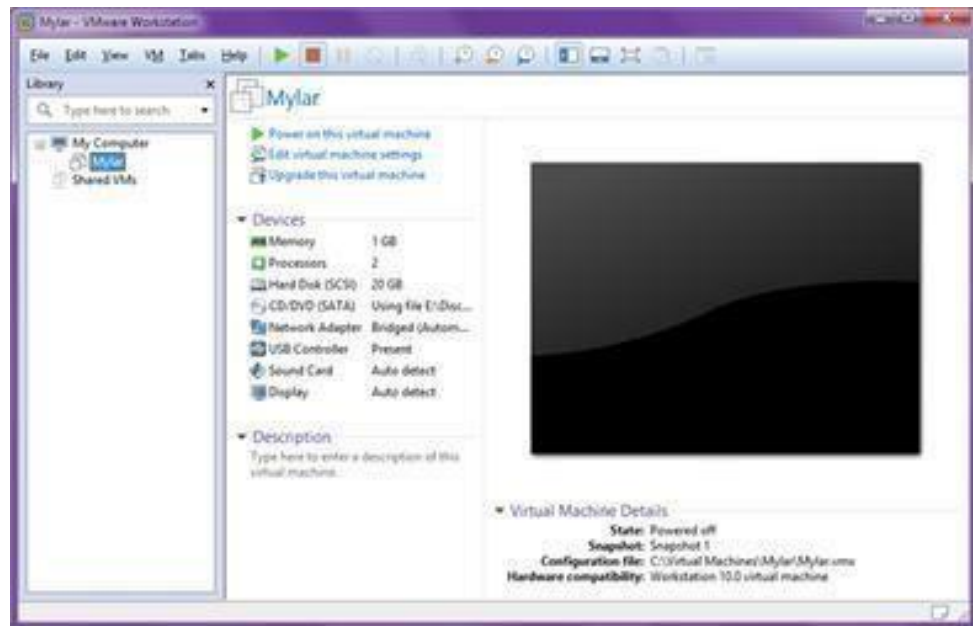
- **Many applications on each server:** As each virtual machine encapsulates an entire machine, many applications and operating systems can run on a single host at the same time.
- **Maximum server utilization, minimum server count:** Every physical machine is used to its full capacity, allowing you to significantly reduce costs by deploying fewer servers overall.
- **Faster, easier application and resource provisioning:** As self-contained software files, virtual machines can be manipulated with copy-and-paste ease. Virtual machines can even be transferred from one physical server to another while running, via a process known as live migration.

5.10.1 Setting up a VMware Workstation

5.10.2 VMware Workstation is developed and sold by VMware, Inc., a division of EMC Corporation. VMware Workstation is a hypervisor that runs on x86 or x86-64 computers; it enables users to set up one or more virtual machines (VMs) on a single physical machine, and use them simultaneously along with the actual machine.

Each virtual machine can execute its own operating system, including versions of Microsoft Windows, Linux, BSD, and MS-DOS. VMware Workstation supports bridging existing host network adapters and share physical disk drives and USB devices with a virtual machine. In addition, it can simulate disk drives. It can mount an existing ISO image file into a virtual optical disc drive so that the virtual machine sees it as a real one. Likewise, virtual hard disk drives are made via **.vmdk** files.

VMware Workstation can save the state of a virtual machine (a "snapshot") at any instant. These snapshots can later be restored, effectively returning the virtual machine to the saved state.



VMware Workstation

VMware Workstation includes the ability to designate multiple virtual machines as a team which can then be powered on, powered off, suspended or resumed as a single object, making it particularly useful for testing client-server environments.

VMware Player

The VMware Player, a virtualization package of basically similar, but reduced, functionality, is also available, and is free of charge for non-commercial use, or for distribution or other use by written agreement.

VMware Player is a virtualization software package supplied free of charge by VMware, Inc. VMware Player can run existing virtual appliances and create its own virtual machines. It uses the same virtualization core as VMware Workstation, a similar program with more features, but not free of charge. VMware Player is available for personal non-commercial use, or for distribution or other use by written agreement.

VMware claims the Player offers better graphics, faster performance, and tighter integration for running Windows XP under Windows Vista or Windows 7 than Microsoft's Windows XP Mode running on Windows Virtual PC, which is free of charge for all purposes.

VMware Tools

VMware Tools is a package with drivers and other software that can be installed in guest operating systems to increase their performance. It has several components, including the following drivers for the emulated hardware:

- VESA-compliant graphics for the guest machine to access high screen resolutions

- Mouse integration, Drag-and-drop file support
- Clipboard sharing between host and guest
- Time synchronization capabilities (guest syncs with host machine's clock)
- Support for Unity, a feature that allows seamless integration of applications with the host desktop

Installing and Configuring VMWare

1. Download VMware Server 2. VMware management console on a remote Ubuntu desktop behind a firewall at a remote location. Run the following command:

```
$gksu vmware-server-console
```

2. Install the VMware Server 2.0.2 rpm as shown below.

```
# rpm -ivh VMware-server-2.0.2-203138.i386.rpm
Preparing...
```

```
1:VMware-server
```

```
##### [100%]
```

The installation of VMware Server 2.0.2 for Linux completed successfully.

You can decide to remove this software from your system at any time by invoking the following command:

```
rpm -e VMware-server
```

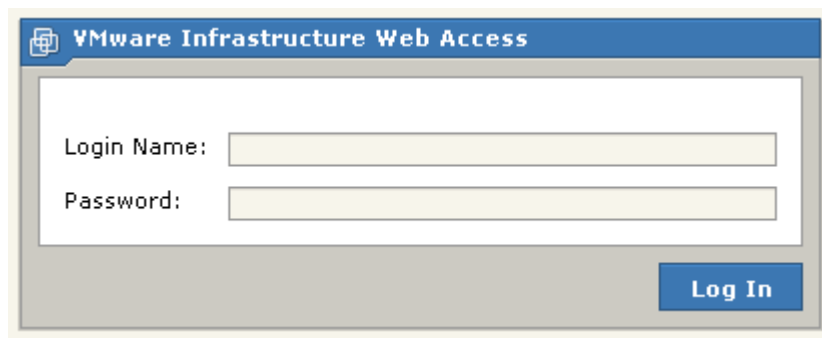
Before running VMware Server for the first time, you need to configure it for your running kernel by invoking the following command:

```
/usr/bin/vmware-config.pl
```

3. Configure VMware Server 2 using *vmware-config.pl*. Execute the *vmware-config.pl* as shown below. Accept default values for everything. Partial output of the *vmware-config.pl* is shown below.

```
# /usr/bin/vmware-config.pl
```

4. Go to VMware Infrastructure Webaccess. Go to **<https://{host-os-ip}:8333/ui>** to access the VMware Infrastructure web access console.



VMware Web Access Login

Installing a VMware Guest OS

1. Start VMware Workstation

Windows host: Double-click the VMware Workstation icon on your desktop or use the Start menu (Start > Programs > VMware > VMware Workstation).

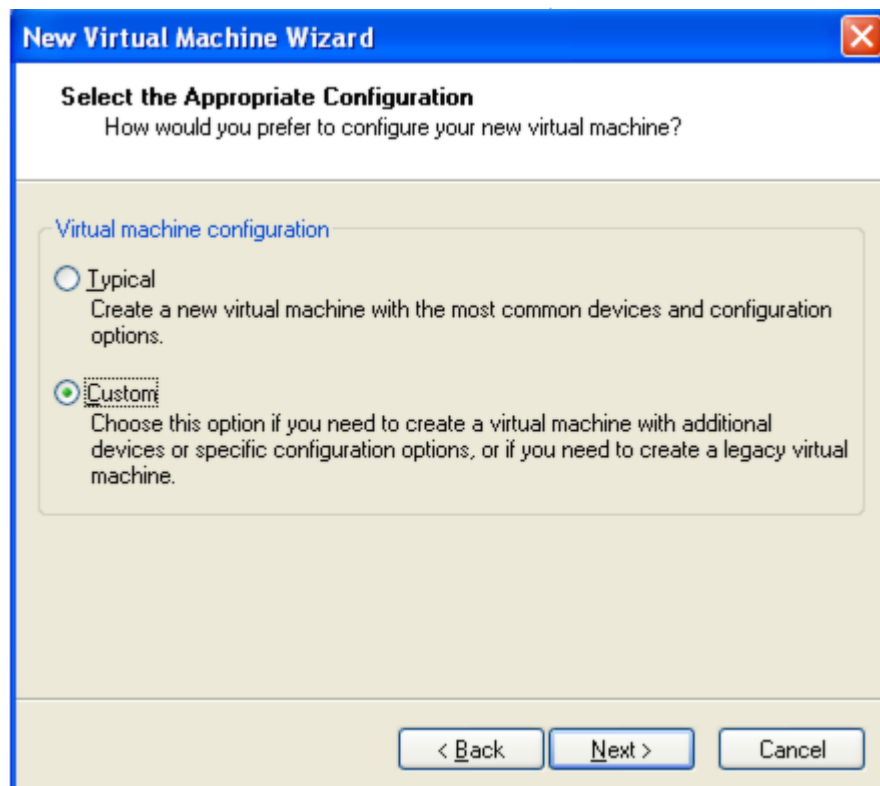
Linux host: In a terminal window, enter the command

```
vmware &
```

2. Start the New Virtual Machine Wizard

When you start VMware Workstation, you can open an existing virtual machine or create a new one. Choose File > New > Virtual Machine to begin creating your virtual machine.

3. Select the method you want to use for configuring your virtual machine.



If you select **Typical**, the wizard prompts you to specify or accept defaults for the following choices:

- The guest operating system
- The virtual machine name and the location of the virtual machine's files
- The network connection type
- Whether to allocate all the space for a virtual disk at the time you create it
- Whether to split a virtual disk into 2GB files

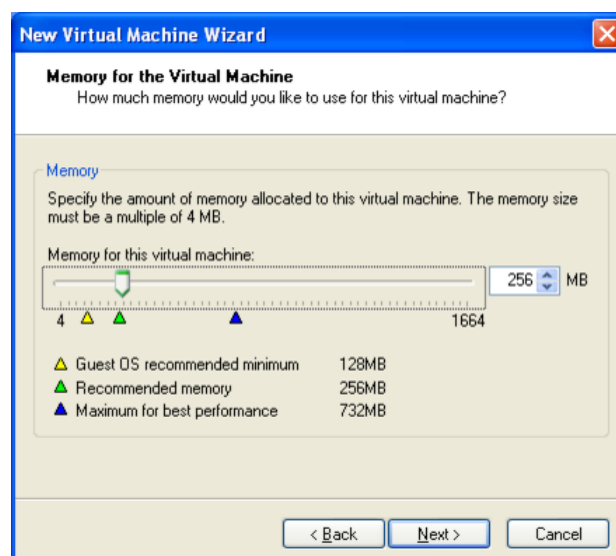
If you select **Custom**, the wizard prompts you to specify or accept defaults for the following choices:

- Make a legacy virtual machine that is compatible with Workstation 4.x, GSX Server 3.x, ESX Server 2.x and VMware ACE 1.x.
 - Use an IDE virtual disk for a guest operating system that would otherwise have a SCSI virtual disk created by default
 - Use a physical disk rather than a virtual disk and Set memory options that are different from the defaults
4. Select a guest operating system and type a name and folder for the virtual machine.

Linux hosts: The default location for this Windows XP Professional virtual machine is `<homedir>/vmware/winXPPro`, where `<homedir>` is the home directory of the user who is currently logged on.



5. Specify the number of processors for the virtual machine. The setting Two is supported only for host machines with at least two logical processors.



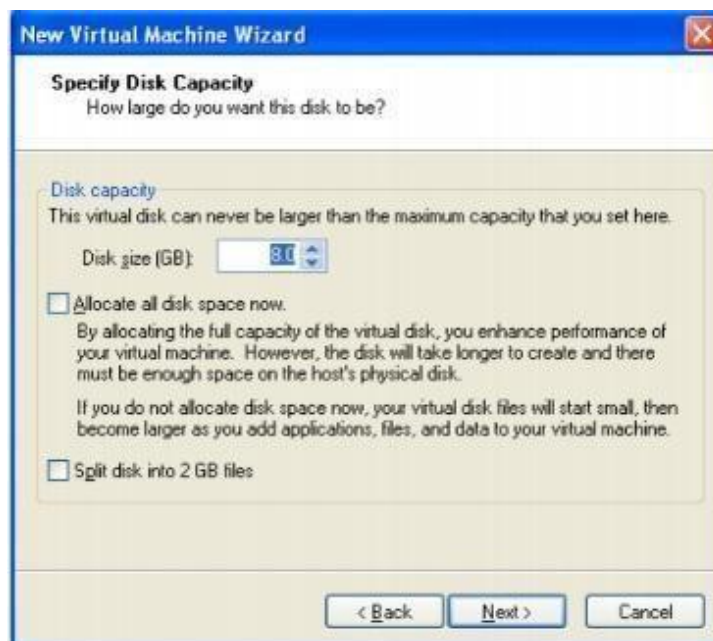
If you selected **Custom** as your configuration path, you may adjust the memory settings or accept the defaults, then click Next to continue.

6. Configure the networking capabilities of the virtual machine.

If you selected *Typical* as your configuration path, click Finish and the wizard sets up the files needed for your virtual machine.

If you selected *Custom* as your configuration path, continue with the steps below to configure a disk for your virtual machine.

7. Select whether to create an IDE or SCSI disk and specify the capacity of the virtual disk.



8. Click Finish. The wizard sets up the files needed for your virtual machine.

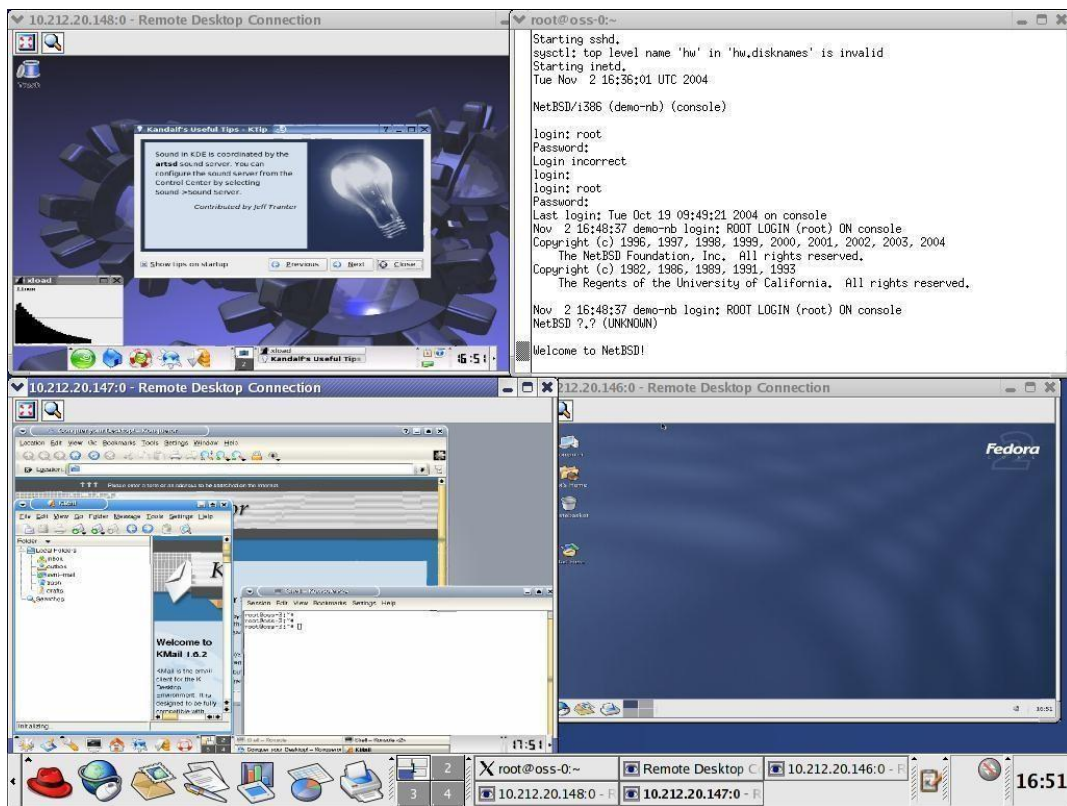
5.10.3 Setting up a XEN Workstation

Xen is a hypervisor using a microkernel design, providing services that allow multiple computer operating systems to execute on the same computer hardware concurrently.

The University of Cambridge Computer Laboratory developed the first versions of Xen. The Xen community develops and maintains Xen as free and open-source software, subject to the requirements of the GNU General Public License (GPL), version 2. Xen is currently available for the IA-32, x86-64 and ARM instruction sets.

XenServer runs directly on server hardware without requiring an underlying operating system, which results in an efficient and scalable system. XenServer works by abstracting elements from the physical machine (such as hard drives, resources

and ports) and allocating
them to the virtual machines running on it.



XEN Environment

Responsibilities of the hypervisor include memory management and CPU scheduling of all virtual machines, and for launching the most privileged domain - the only virtual machine which by default has direct access to hardware. From the dom0 the hypervisor can be managed and unprivileged domains can be launched.

Benefits of Using XenServer

1. Using XenServer reduces costs by:

- Consolidating multiple VMs onto physical servers
- Reducing the number of separate disk images that need to be managed
- Allowing for easy integration with existing networking and storage infrastructures

2. Using XenServer increases flexibility by:

- Allowing you to schedule zero downtime maintenance by using XenMotion to live migrate VMs between XenServer hosts
- Increasing availability of VMs by using High Availability to configure policies that restart VMs on another XenServer host if one fails
- Increasing portability of VM images, as one VM image will work on a range of deployment infrastructures

Administering XenServer

- There are two methods by which to administer XenServer: XenCenter and the XenServer Command-Line Interface (CLI).
- **XenCenter** is a graphical, Windows-based user interface. XenCenter allows you to manage XenServer hosts, pools and shared storage, and to deploy, manage and monitor VMs from your Windows desktop machine.
- The XenCenter on-line Help is a useful resource for getting started with XenCenter and for context-sensitive assistance.

Installing and Configuring XenServer

1. Type the following command to get information about xen server package

```
# yum info xen
```

2. Run the system-config-securitylevel program or edit /etc/selinux/config to looks as follows:

```
SELINUX=Disabled
SELINUXTYPE=targeted
```

If you changed the SELINUX value from enforcing, you'll need to reboot Fedora before proceeding.

3. This command will install the Xen hypervisor, a Xen-modified Fedora kernel called *domain 0*, and various utilities:

```
# yum install kernel-xen0
```

4. To make the Xen kernel the default, change this line:

```
default=1
```

to

```
default=0
```

5. Now you can reboot. Xen should start automatically, but let's check:

```
# /usr/sbin/xm list
```

Name	ID	Mem (MiB)	VCPUs	State	Time (s)
Domain-0	0	880	1	r-----	20.5

The output should show that Domain-0 is running. Domain 0 controls all the guest operating systems that run on the processor, similarly to how the kernel controls processes in an operating system.

Installing a Xen Guest OS from the Command-line

1. Preparing the System for virt-install

Fedora Linux does not install VNC by default. To verify whether VNC is installed, run the following command from a Terminal Window:

If rpm reports that VNC is not installed, it may be installed from root as follows:

```
yum install vnc
```

2. Running virt-install to Build the Xen Guest System

virt-install must be run as root and, once invoked, will ask a number of questions before creating the guest system. The questions are as follows:

- i. What is the name of your virtual machine and install location?
- ii. How much RAM should be allocated (in megabytes)?
- iii. What would you like to use as the disk (path)?
- iv. Would you like to enable graphics support? (yes or no)

The following transcript shows a typical virt-install session:

```
# virt-install
```

3. Once the guest system has been created, the vncviewer screen will appear containing the operating system installer:



Installing a Xen Guest OS (Fedora Core 5)

1. Fedora Core 5 has a Xen guest installation script that simplifies the process, although it installs only FC5 guests. The script expects to access the FC5 install tree via FTP, the Web, or NFS; for some reason, you can't specify a directory or file.

```
# mkdir /var/www/html/dvd
# mount -t iso9660 /dev/dvd /var/www/html/dvd
# apachectl start
```

Now we'll run the installation script and answer its questions:

```
# xenguest-install.py
```

2. Xen does not start the guest operating system automatically. You need to type this command on the host:

3. To prove that both servers are running, try these commands:

```
# xm list
# xentop
```

4. To start Xen domains automatically, use these commands:

```
# /sbin/chkconfig --level 345 xendomains on
# /sbin/service xendomains start
```

5. To Edit A Xen Guest Configuration File, Which Is A Text File (Actually, A Python Script) In The

/Etc/Xen Directory.

```
# man xmdomain.cfg
```

And edit as follows,

```
# Automatically generated Xen config file
name = "guest1"
memory = "256"
disk = [ 'file:/xenguest,xvda,w' ]
vif = [ 'mac=00:16:3e:63:c7:76' ]
uuid = "bc2c1684-c057-99ea-962b-de44a038bbda"
bootloader="/usr/bin/pygrub"
on_reboot = 'restart'
on_crash = 'restart'
```

6. Once you have a guest configuration file, create the Xen guest with this command:

```
# xm create -c guest_name
```

where

guest_name can be a full pathname or a relative filename (in which case Xen places

it in /etc/xen/guest_name).

Xen will create the guest domain and try to boot it from the given file or device. The **-c** option attaches a console to the domain when it starts, so you can answer the installation questions that appear .

CS3301-DATA STRUCTURES QUESTION BANK

UNIT I

2MARKS

1. Define data structure.

The data structure can be defined as the collection of elements and all the possible operations which are required for those set of elements. Formally data structure can be defined as a data structure is a set of domains D, a set of domains F and a set of axioms A. this triple (D,F,A) denotes the data structure d.

2. What do you mean by non-linear data structure? Give example.

The non-linear data structure is the kind of data structure in which the data may be arranged in hierarchical fashion. For example- Trees and graphs.

3. What do you linear data structure? Give example.

The linear data structure is the kind of data structure in which the data is linearly arranged. For example- stacks, queues, linked list.

4. List the various operations that can be performed on data structure.

Various operations that can be performed on the data structure are

- Create
- Insertion of element
- Deletion of element
- Searching for the desired element
- Sorting the elements in the data structure
- Reversing the list of elements.

5. What is abstract data type? What are all not concerned in an ADT?

The abstract data type is a triple of D i.e. set of axioms, F-set of functions and A-Axioms in which only what is to be done is mentioned but how is to be done is not mentioned. Thus ADT is not concerned with implementation details.

6. List out the areas in which data structures are applied extensively.

Following are the areas in which data structures are applied extensively.

- Operating system- the data structures like priority queues are used for scheduling the jobs in the operating system.
- Compiler design- the tree data structure is used in parsing the source program. Stack data structure is used in handling recursive calls.

- Database management system- The file data structure is used in database management systems. Sorting and searching techniques can be applied on these data in the file.
- Numerical analysis package- the array is used to perform the numerical analysis on the given set of data.
- Graphics- the array and the linked list are useful in graphics applications.
- Artificial intelligence- the graph and trees are used for the applications like building expression trees, game playing.

7. What is a linked list?

A linked list is a set of nodes where each node has two fields 'data' and 'link'. The data field is used to store actual piece of information and link field is used to store address of next node.

8. What are the pitfall encountered in singly linked list?

Following are the pitfall encountered in singly linked list

- The singly linked list has only forward pointer and no backward link is provided. Hence the traversing of the list is possible only in one direction. Backward traversing is not possible.
- Insertion and deletion operations are less efficient because for inserting the element at desired position the list needs to be traversed. Similarly, traversing of the list is required for locating the element which needs to be deleted.

9. Define doubly linked list.

Doubly linked list is a kind of linked list in which each node has two link fields. One link field stores the address of previous node and the other link field stores the address of the next node.

10. Write down the steps to modify a node in linked lists.

- Enter the position of the node which is to be modified.
- Enter the new value for the node to be modified.
- Search the corresponding node in the linked list.
- Replace the original value of that node by a new value.
- **Display the messages as "The node is modified".**

11. Difference between arrays and lists.

In arrays any element can be accessed randomly with the help of index of array, whereas in lists any element can be accessed by sequential access only.

Insertion and deletion of data is difficult in arrays on the other hand insertion and deletion of data is easy in lists.

12. State the properties of LIST abstract data type with suitable example.

Various properties of LIST abstract data type are

- (i) It is linear data structure in which the elements are arranged adjacent to each other.
- (ii) It allows to store single variable polynomial.
- (iii) If the LIST is implemented using dynamic memory then it is called linked list.

Example of LIST are- stacks, queues, linked list.

13. State the advantages of circular lists over doubly linked list.

In circular list the next pointer of last node points to head node, whereas in doubly linked list each node has two pointers: one previous pointer and another is next pointer. The main advantage of circular list over doubly linked list is that with the help of single pointer field we can access head node quickly. Hence some amount of memory get saved because in circular list only one pointer is reserved.

14. What are the advantages of doubly linked list over singly linked list?

The doubly linked list has two pointer fields. One field is previous link field and another is next link field. Because of these two pointer fields we can access any node efficiently whereas in singly linked list only one pointer field is there which stores forward pointer.

15. Why is the linked list used for polynomial arithmetic?

We can have separate coefficient and exponent fields for representing each term of polynomial. Hence there is no limit for exponent. We can have any number as an exponent.

16. What is the advantage of linked list over arrays?

The linked list makes use of the dynamic memory allocation. Hence the user can allocate or de allocate the memory as per his requirements. On the other hand, the array makes use of the static memory location. Hence there are chances of wastage of the memory or shortage of memory for allocation.

17. What is the circular linked list?

The circular linked list is a kind of linked list in which the last node is connected to the first node or head node of the linked list.

18. What is the basic purpose of header of the linked list?

The header node is the very first node of the linked list. Sometimes a dummy value such - 999 is stored in the data field of header node.

This node is useful for getting the starting address of the linked list.

19. What is the advantage of an ADT?

- **Change:** the implementation of the ADT can be changed without making changes in the client program that uses the ADT.
- **Understandability:** ADT specifies what is to be done and does not specify the implementation details. Hence code becomes easy to understand due to ADT.
- **Reusability:** the ADT can be reused by some program in future.

20. What is static linked list? State any two applications of it.

- The linked list structure which can be represented using arrays is called static linked list.
- It is easy to implement, hence for creation of small databases, it is useful
- The searching of any record is efficient, hence the applications in which the record need to be searched quickly, the static linked list are used.

16 MARKS

1. Explain the insertion operation in linked list. How nodes are inserted after a specified node.
2. Write an algorithm to insert a node at the beginning of list?
3. Discuss the merge operation in circular linked lists.
4. What are the applications of linked list in dynamic storage management?
5. How polynomial expression can be represented using linked list?
6. What are the benefit and limitations of linked list?
7. Define the deletion operation from a linked list.
8. What are the different types of data structure?
9. Explain the operation of traversing linked list. Write the algorithm and give an example.

UNIT II

2MARKS

1. Define Stack

A Stack is an ordered list in which all insertions (Push operation) and deletion (Pop operation) are made at one end, called the top. The topmost element is pointed by top. The top is initialized to -1 when the stack is created that is when the stack is empty. In a stack $S = (a_1, a_n)$, a_1 is the bottom most element and element a_i is on top of element a_{i-1} . Stack is also referred as Last In First Out (LIFO) list.

2. What are the various Operations performed on the Stack?

The various operations that are performed on the stack are

CREATE(S) – Creates S as an empty stack.

PUSH(S,X) – Adds the element X to the top of the stack.

POP(S) – Deletes the top most elements from the stack.

TOP(S) – returns the value of top element from the stack.

ISEMPTY(S) – returns true if Stack is empty else false.

ISFULL(S) - returns true if Stack is full else false.

3. Write the postfix form for the expression $-A+B-C+D$?

$A-B+C-D+$

4. What are the postfix and prefix forms of the expression?

$A+B*(C-D)/(P-R)$

Postfix form: $ABCD-*PR-/+$

Prefix form: $+A/*B-CD-PR$

5. Explain the usage of stack in recursive algorithm implementation?

In recursive algorithms, stack data structures is used to store the return address when a recursive call is encountered and also to store the values of all the parameters essential to the current state of the function.

6. Define Queue.

A Queue is an ordered list in which all insertions take place at one end called the rear, while all deletions take place at the other end called the front. Rear is initialized to -1 and front is initialized to 0. Queue is also referred as First In First Out (FIFO) list.

7. What are the various operations performed on the Queue?

The various operations performed on the queue are

CREATE(Q) – Creates Q as an empty Queue.

Enqueue(Q,X) – Adds the element X to the Queue.

Dequeue(Q) – Deletes a element from the Queue.

ISEMPTY(Q) – returns true if Queue is empty else false.

ISFULL(Q) - returns true if Queue is full else false.

8. How do you test for an empty Queue?

The condition for testing an empty queue is $\text{rear} = \text{front} - 1$. In linked list implementation of queue the condition for an empty queue is the header node link field is NULL.

9. Write down the function to insert an element into a queue, in which the queue is implemented as an array. (May 10)

Q – Queue

X – element to added to the queue Q IsFull(Q)

– Checks and true if Queue Q is full $Q \rightarrow \text{Size} -$

Number of elements in the queue Q $Q \rightarrow \text{Rear} -$

Points to last element of the queue Q $Q \rightarrow \text{Array}$

– array used to store queue elements void

```
enqueue (int X, Queue Q) {
```

```
    if(IsFull(Q))
```

```
        Error (“Full queue”);
```

```
    else    {
```

```
        Q->Size++;
```

```

        Q->Rear = Q->Rear+1;
        Q->Array[ Q->Rear ]=X;
    }
}

```

10..Define Dequeue.

Deque stands for Double ended queue. It is a linear list in which insertions and deletion are made from either end of the queue structure.

11. Define Circular Queue.

Another representation of a queue, which prevents an excessive use of memory by arranging elements/ nodes Q_1, Q_2, \dots, Q_n in a circular fashion. That is, it is the queue, which wraps around upon reaching the end of the queue

12. List any four applications of stack.

- Parsing context free languages
- Evaluating arithmetic expressions
- Function call
- Traversing trees and graph
- Tower of Hanoi

16 MARKS

1. Write an algorithm for Push and Pop operations on Stack using Linked list. (8)
2. Explain the linked list implementation of stack ADT in detail?
3. Define an efficient representation of two stacks in a given area of memory with n words and explain.
4. Explain linear linked implementation of Stack and Queue?
 - a. Write an ADT to implement stack of size N using an array. The elements in the stack are to be integers. The operations to be supported are PUSH, POP and DISPLAY. Take into account the exceptions of stack overflow and stack underflow. (8)
 - b. A circular queue has a size of 5 and has 3 elements 10,20 and 40 where $F=2$ and $R=4$. After inserting 50 and 60, what is the value of F and R. Trying to insert 30 at this stage what happens? Delete 2 elements from the queue and insert 70, 80 & 90. Show the sequence of steps with necessary diagrams with the value of F & R. (8 Marks)
5. Write the algorithm for converting infix expression to postfix (polish) expression?

6. Explain in detail about priority queue ADT in detail?
7. **Write a function called 'push' that takes two parameters: an integer variable and a stack into** which it would push this element and returns a 1 or a 0 to show success of addition or failure.
8. What is a DeQueue? Explain its operation with example?
9. Explain the array implementation of queue ADT in detail?
10. Explain the addition and deletion operations performed on a circular queue with necessary algorithms.(8) (Nov 09)

UNIT III

2MARKS

1. Define tree?

Trees are non-linear data structure, which is used to store data items in a sorted sequence. It represents any hierarchical relationship between any data item. It is a collection of nodes, which has a distinguished node called the root and zero or more non-empty subtrees T_1, T_2, \dots, T_k , each of which are connected by a directed edge from the root.

2. Define Height of tree?

The height of n is the length of the longest path from root to a leaf. Thus all leaves have height zero. The height of a tree is equal to a height of a root.

3. Define Depth of tree?

For any node n , the depth of n is the length of the unique path from the root to node n . Thus for a root the depth is always zero.

4. What is the length of the path in a tree?

The length of the path is the number of edges on the path. In a tree there is exactly one path from the root to each node.

5. Define sibling?

Nodes with the same parent are called siblings. The nodes with common parents are called siblings.

6. Define binary tree?

A Binary tree is a finite set of data items which is either empty or consists of a single item called root and two disjoint binary trees called left subtree. The maximum degree of any node is two.

7. What are the two methods of binary tree implementation?

Two methods to implement a binary tree are,

a. Linear representation.

b. Linked representation

8. What are the applications of binary tree?

Binary tree is used in data processing.

a. File index schemes

b. Hierarchical database management system

9. List out few of the Application of tree data-structure?

- Ø The manipulation of Arithmetic expression
- Ø Used for Searching Operation
- Ø Used to implement the file system of several popular operating systems
- Ø Symbol Table construction
- Ø Syntax analysis

10. Define expression tree?

Expression tree is also a binary tree in which the leafs terminal nodes or operands and non-terminal intermediate nodes are operators used for traversal.

11. Define tree- traversal and mention the type of traversals?

Visiting of each and every node in the tree exactly is called as tree traversal.

Three types of tree traversal

1. Inorder traversal
2. Preoder traversal
3. Postorder traversal.

12. Define in -order traversal?

In-order traversal entails the following steps;

- a. Traverse the left subtree
- b. Visit the root node
- c. Traverse the right subtree

13. Define threaded binary tree.

A binary tree is threaded by making all right child pointers that would normally be null point to the in order successor of the node, and all left child pointers that would normally be null point to the in order predecessor of the node.

14. What are the types of threaded binary tree?

- i. Right-in threaded binary tree
- ii. Left-in threaded binary tree
- iii. Fully-in threaded binary tree

15. Define Binary Search Tree.

Binary search tree is a binary tree in which for every node X in the tree, the values of all the keys in its left subtree are smaller than the key value in X and the values of all the keys in its right subtree are larger than the key value in X.

16. What is AVL Tree?

AVL stands for Adelson-Velskii and Landis. An AVL tree is a binary search tree which has the following properties:

1. The sub-trees of every node differ in height by at most one.
2. Every sub-tree is an AVL tree.

Search time is $O(\log n)$. Addition and deletion operations also take $O(\log n)$ time.

17. List out the steps involved in deleting a node from a binary search tree.

- Deleting a node is a leaf node (ie) No children
- Deleting a node with one child.
- Deleting a node with two Childs.

18. What is 'B' Tree?

A B-tree is a tree data structure that keeps data sorted and allows searches, insertions, and deletions in logarithmic amortized time. Unlike self-balancing binary search trees, it is optimized for systems that read and write large blocks of data. It is most commonly used in database and file systems.

19. Define complete binary tree.

If all its levels, possible except the last, have maximum number of nodes and if all the nodes in the last level appear as far left as possible

16 MARKS

1. Explain the AVL tree insertion and deletion with suitable example.
2. Describe the algorithms used to perform single and double rotation on AVL tree.
3. Explain about B-Tree with suitable example.
4. Explain about B+ trees with suitable algorithm.
5. Write short notes on
 - i. Binomial heaps
 - ii. Fibonacci heaps
6. Explain the tree traversal techniques with an example.
7. Construct an expression tree for the expression $(a+b*c) + ((d*e+f)*g)$. Give the outputs when you apply inorder, preorder and postorder traversals.
8. How to insert and delete an element into a binary search tree and write down the code for the insertion routine with an example.
9. What are threaded binary tree? Write an algorithm for inserting a node in a threaded binary tree.
10. Create a binary search tree for the following numbers start from an empty binary search tree. 45,26,10,60,70,30,40 Delete keys 10,60 and 45 one after the other and show the trees at each stage.

UNIT- IV

2.MARKS

1. Write the definition of weighted graph?

A graph in which weights are assigned to every edge is called a weighted graph.

2. Define Graph?

A graph G consist of a nonempty set V which is a set of nodes of the graph, a set E which is the set of edges of the graph, and a mapping from the set of edges E to set of pairs of elements of V . It can also be represented as $G=(V, E)$.

3. Define adjacency matrix?

The adjacency -matrix is an $n \times n$ matrix A whose elements a_{ij} are given by $a_{ij} = 1$ if (v_i, v_j) Exists $=0$ otherwise

4. Define adjacent nodes?

Any two nodes, which are connected by an edge in a graph, are called adjacent nodes. For example, if an edge $x \in E$ is associated with a pair of nodes

(u, v) where $u, v \in V$, then we say that the edge x connects the nodes u and v .

5. What is a directed graph?

A graph in which every edge is directed is called a directed graph.

6. What is an undirected graph?

A graph in which every edge is undirected is called an undirected graph.

7. What is a loop?

An edge of a graph, which connects to itself, is called a loop or sling.

8. What is a simple graph?

A simple graph is a graph, which has not more than one edge between a pair of nodes.

9. What is a weighted graph?

A graph in which weights are assigned to every edge is called a weighted graph.

10. Define indegree and out degree of a graph?

In a directed graph, for any node v , the number of edges, which have v as their initial node, is called the out degree of the node v .

Outdegree: Number of edges having the node v as root node is the outdegree of the node v .

11. Define path in a graph?

The path in a graph is the route taken to reach terminal node from a starting node.

12. What is a simple path?

- i. A path in a diagram in which the edges are distinct is called a simple path.
- ii. It is also called as edge simple.

13. What is a cycle or a circuit?

A path which originates and ends in the same node is called a cycle or circuit.

14. What is an acyclic graph?

A simple diagram, which does not have any cycles, is called an acyclic graph.

15. What is meant by strongly connected in a graph?

An undirected graph is connected, if there is a path from every vertex to every other vertex. A directed graph with this property is called strongly connected.

16. When a graph said to be weakly connected?

$a_{ij} = 1$ if (v_i, v_j) Exists $= 0$ otherwise

When a directed graph is not strongly connected but the underlying graph is connected, then the graph is said to be weakly connected.

17. Name the different ways of representing a graph? Give examples (Nov 10)

- a. Adjacency matrix
- b. Adjacency list

18. What is an undirected acyclic graph?

When every edge in an acyclic graph is undirected, it is called an undirected acyclic graph. It is also called as undirected forest.

19. What is meant by depth?

The depth of a list is the maximum level attributed to any element with in the list or with in any sub list in the list.

20. What is the use of BFS?

BFS can be used to find the shortest distance between some starting node and the remaining nodes of the graph. The shortest distance is the minimum number of edges traversed in order to travel from the start node the specific node being examined.

21. What is topological sort?

It is an ordering of the vertices in a directed acyclic graph, such that: If there is a path from u to v, then v appears after u in the ordering.

22. Write BFS algorithm

- 1. Initialize the first node's dist number and place in queue
- 2. Repeat until all nodes have been examined

3. Remove current node to be examined from queue
4. Find all unlabeled nodes adjacent to current node
5. If this is an unvisited node label it and add it to the queue
6. Finished.

23. Define biconnected graph?

A graph is called biconnected if there is no single node whose removal causes the graph to break into two or more pieces. A node whose removal causes the graph to become disconnected is called a cut vertex.

24. What are the two traversal strategies used in traversing a graph?

- a. Breadth first search
- b. Depth first search

25. Articulation Points (or Cut Vertices) in a Graph

A vertex in an undirected connected graph is an articulation point (or cut vertex) if removing it (and edges through it) disconnects the graph. Articulation points represent vulnerabilities in a connected network – single points whose failure would split the network into 2 or more disconnected components. They are useful for designing reliable networks.

16 MARKS

1. Explain the various representation of graph with example in detail?
2. Explain Breadth First Search algorithm with example?
3. Explain Depth first and breadth first traversal?
4. What is topological sort? Write an algorithm to perform topological sort?(8) (Nov 09)
5. (i) write an algorithm to determine the biconnected components in the given graph. (10) (may 10)
(ii)determine the biconnected components in a graph. (6)
6. Explain the various applications of Graphs.

UNIT – V

2 MARKS

1. What is meant by Sorting?

Sorting is ordering of data in an increasing or decreasing fashion according to some linear relationship among the data items.

2. List the different sorting algorithms.

- Bubble sort
- Selection sort
- Insertion sort
- Shell sort
- Quick sort
- Radix sort
- Heap sort
- Merge sort

3. Why bubble sort is called so?

The bubble sort gets its name because as array elements are sorted they gradually **“bubble” to their proper positions, like bubbles rising** in a glass of soda.

4. State the logic of bubble sort algorithm.

The bubble sort repeatedly compares adjacent elements of an array. The first and second elements are compared and swapped if out of order. Then the second and third elements are compared and swapped if out of order. This sorting process continues until the last two elements of the array are compared and swapped if out of order.

5. What number is always sorted to the top of the list by each pass of the Bubble sort algorithm?

Each pass through the list places the next largest value in its proper place. In essence, each item **“bubbles” up to the location where it belongs.**

6. When does the Bubble Sort Algorithm stop?

The bubble sort stops when it examines the entire array and finds that no "swaps" are needed. The bubble sort keeps track of the occurring swaps by the use of a flag.

7. State the logic of selection sort algorithm.

It finds the lowest value from the collection and moves it to the left. This is repeated until the complete collection is sorted.

8. What is the output of selection sort after the 2nd iteration given the following sequence?

16 3 46 9 28 14

Ans: 3 9 46 16 28 14

9. How does insertion sort algorithm work?

In every iteration an element is compared with all the elements before it. While comparing if it is found that the element can be inserted at a suitable position, then space is created for it by shifting the other elements one position up and inserts the desired element at the suitable position. This procedure is repeated for all the elements in the list until we get the sorted elements.

10. What operation does the insertion sort use to move numbers from the unsorted section to the sorted section of the list?

The Insertion Sort uses the swap operation since it is ordering numbers within a single list.

11. How many key comparisons and assignments an insertion sort makes in its worst case?

The worst case performance in insertion sort occurs when the elements of the input array are in descending order. In that case, the first pass requires one comparison, the second pass **requires two comparisons, third pass three comparisons,....kth pass requires (k-1)**, and finally the last pass requires (n-1) comparisons. Therefore, total numbers of comparisons are:

$$f(n) = 1+2+3+\dots+(n-k)+\dots+(n-2)+(n-1) = n(n-1)/2 = O(n^2)$$

12. Which sorting algorithm is best if the list is already sorted? Why?

Insertion sort as there is no movement of data if the list is already sorted and complexity is of the order O(N).

13. Which sorting algorithm is easily adaptable to singly linked lists? Why?

Insertion sort is easily adaptable to singly linked list. In this method there is an array link of pointers, one for each of the original array elements. Thus the array can be thought of as a linear link list pointed to by an external pointer first initialized to 0. To insert the kth element the linked list is traversed until the proper position for x[k] is found, or until the end of the list is

reached. At that point $x[k]$ can be inserted into the list by merely adjusting the pointers without shifting any elements in the array which reduces insertion time.

14. Why Shell Sort is known diminishing increment sort?

The distance between comparisons decreases as the sorting algorithm runs until the last phase in which adjacent elements are compared. In each step, the sortedness of the sequence is increased, until in the last step it is completely sorted.

15. Which of the following sorting methods would be especially suitable to sort a list L consisting of a sorted list followed by a few “random” elements?

Quick sort is suitable to sort a list L consisting of a sorted list followed by a few “random” elements.

**16. What is the output of quick sort after the 3rd iteration given the following sequence?
24 56 47 35 10 90 82 31**

Pass 1:- (10) 24 (56 47 35 90 82 31)

Pass 2:- 10 24 (56 47 35 90 82 31)

Pass 3:- 10 24 (47 35 31) 56 (90 82)

17. Mention the different ways to select a pivot element.

The different ways to select a pivot element are

- Pick the first element as pivot
- Pick the last element as pivot
- Pick the Middle element as pivot
- Median-of-three elements
- Pick three elements, and find the median x of these elements
- Use that median as the pivot.
- Randomly pick an element as pivot.

18. What is divide-and-conquer strategy?

- Divide a problem into two or more sub problems
- Solve the sub problems recursively
- Obtain solution to original problem by combining these solutions

19. Compare quick sort and merge sort.

Quicksort has a best-case linear performance when the input is sorted, or nearly sorted. It has a worst-case quadratic performance when the input is sorted in reverse, or nearly sorted in reverse.

Merge sort performance is much more constrained and predictable than the performance of quicksort. The price for that reliability is that the average case of merge sort is slower than the average case of quicksort because the constant factor of merge sort is larger.

20. Define Searching.

Searching for data is one of the fundamental fields of computing. Often, the difference between a fast program and a slow one is the use of a good algorithm for the data set. Naturally, the use of a hash table or binary search tree will result in more efficient searching, but more often than not an array or linked list will be used. It is necessary to understand good ways of searching data structures not designed to support efficient search.

21. What is linear search?

In Linear Search the list is searched sequentially and the position is returned if the key element to be searched is available in the list, otherwise -1 is returned. The search in Linear Search starts at the beginning of an array and move to the end, testing for a match at each item.

22. What is Binary search?

A binary search, also called a dichotomizing search, is a digital scheme for locating a specific object in a large set. Each object in the set is given a key. The number of keys is always a power of 2. If there are 32 items in a list, for example, they might be numbered 0 through 31 (binary 00000 through 11111). If there are, say, only 29 items, they can be numbered 0 through 28 (binary 00000 through 11100), with the numbers 29 through 31 (binary 11101, 11110, and 11111) as dummy keys.

23. Define hash function?

Hash function takes an identifier and computes the address of that identifier in the hash table using some function.

16 MARKS

1. Write an algorithm to implement Bubble sort with suitable example.
2. Explain any two techniques to overcome hash collision.
3. Write an algorithm to implement insertion sort with suitable example.
4. Write an algorithm to implement selection sort with suitable example.
5. Write an algorithm to implement radix sort with suitable example.
6. Write an algorithm for binary search with suitable example.
7. Discuss the common collision resolution strategies used in closed hashing system.
8. Given the input { 4371, 1323, 6173, 4199, 4344, 9679, 1989 } and a hash function of $h(X)=X(\text{mod } 10)$ show the resulting:
 - a. Separate Chaining hash table
 - b. Open addressing hash table using linear probing
9. Explain Re-hashing and Extendible hashing.

10. Show the result of inserting the keys 2,3,5,7,11,13,15,6,4 into an initially empty extendible hashing data structure with $M=3$. (8) (Nov 10)

11. what are the advantages and disadvantages of various collision resolution strategies? (6)



MADHA
Expertise | Empathy | Excellence
ENGINEERING COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COMMON FOR: DEPARTMENT OF INFORMATION TECHNOLOGY

CS3351 – DIGITAL PRINCIPLES AND COMPUTER ORGANIZATION

YEAR / SEM : II / III

R – 2021

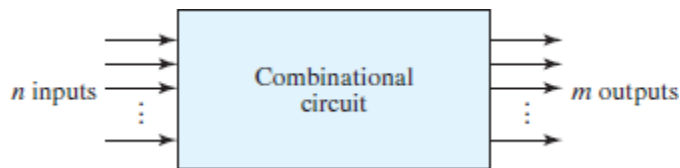
LECTURE NOTES

UNIT I COMBINATIONAL LOGIC

Combinational Circuits – Analysis and Design Procedures - Binary Adder- Subtractor -Decimal Adder - Binary Multiplier - Magnitude Comparator - Decoders – Encoders – Multiplexers - Introduction to HDL – HDL Models of Combinational circuits.

COMBINATIONAL CIRCUITS

- ❖ *A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.*
- ❖ *A combinational circuit performs an operation that can be specified logically by a set of Boolean functions.*



Sequential circuits:

- ❖ *Sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements.*
- ❖ *Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states.*

ANALYSIS PROCEDURE

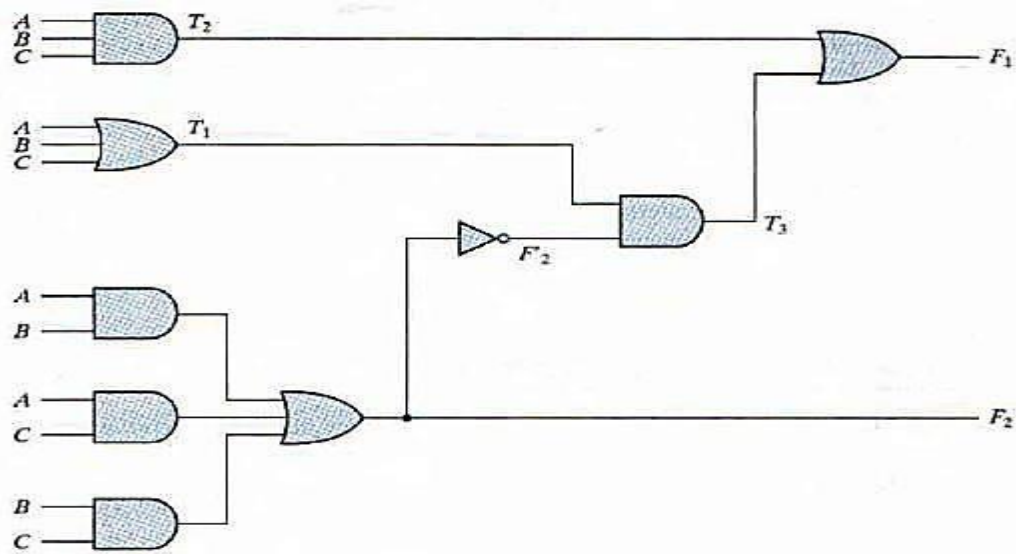
Explain the analysis procedure. Analyze the combinational circuit the following logic diagram.

(May

2015)

- ❖ The analysis of a combinational circuit requires that we determine the function that the circuit implements.
- ❖ The analysis can be performed manually by finding the Boolean functions or truth table or by using a computer simulation program.
- ❖ The first step in the analysis is to make that the given circuit is combinational or sequential.
- ❖ Once the logic diagram is verified to be combinational, one can proceed to obtain the output Boolean functions or the truth table.
- ❖ To obtain the output Boolean functions from a logic diagram,
 - ✓ Label all gate outputs that are a function of input variables with arbitrary symbols or names. Determine the Boolean functions for each gate output.
 - ✓ Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols or names. Find the Boolean functions for these gates.
 - ✓ Repeat the process in step 2 until the outputs of the circuit are obtained.
 - ✓ By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

Logic diagram for analysis example



The Boolean functions for the above outputs are,

$$F_2 = AB + AC + BC$$

$$T_1 = A + B + C$$

$$T_2 = ABC$$

Next, we consider outputs of gates that are a function of already defined symbols:

$$T_3 = F_2 T_1$$

$$F_1 = T_3 + T_2$$

To obtain F_1 as a function of A , B , and C , we form a series of substitutions as follows:

$$\begin{aligned} F_1 &= T_3 + T_2 = F_2 T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

- ❖ Proceed to obtain the truth table for the outputs of those gates which are a function of previously defined values until the columns for all outputs are determined.

Truth Table for the Logic Diagram

A	B	C	F₂	F₂	T₁	T₂	T₃	F₁
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

DESIGNPROCEDURE

Explain the procedure involved in designing combinational circuits.

- ❖ The design of combinational circuits starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained.
- ❖ The procedure involved involves the following steps,
 - ✓ From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
 - ✓ Derive the truth table that defines the required relationship between inputs and outputs.
 - ✓ Obtain the simplified Boolean functions for each output as a function of the input variables.
 - ✓ Draw the logic diagram and verify the correctness of the design.

CIRCUITS FOR ARITHMETIC OPERATIONS

Half adder:

Construct a half adder with necessary diagrams.

(Nov-06, May- 07)

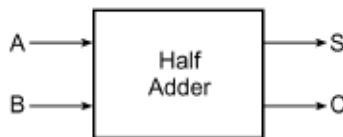
- ❖ A half-adder is an arithmetic circuit block that can be used to add two bits and produce two outputs SUM and CARRY.
- ❖ The Boolean expressions for the SUM and CARRY outputs are given by the equations

$$\text{SUM } S = A.\bar{B} + \bar{A}.B$$

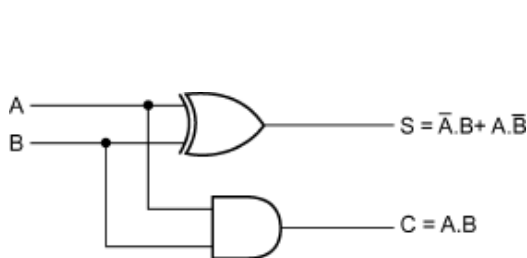
$$\text{CARRY } C = A.B$$

Truth Table:

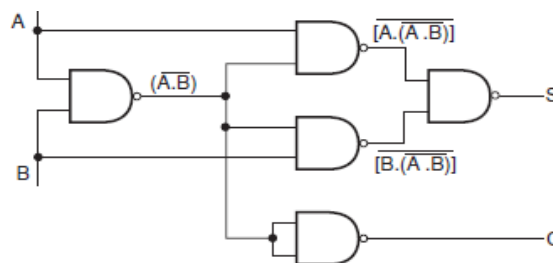
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Logic Diagram:



Half adder using NAND gate:



Full adder:

Design a full adder using NAND and NOR gates respectively.

(Nov -10)

- ❖ A Full-adder is an arithmetic circuit block that can be used to add three bits and produce two outputs SUM and CARRY.
- ❖ The Boolean expressions for the SUM and CARRY outputs are given by the equations

$$S = \bar{A}.\bar{B}.C_{in} + \bar{A}.B.\bar{C}_{in} + A.\bar{B}.\bar{C}_{in} + A.B.C_{in}$$

$$C_{out} = B.C_{in} + A.B + A.C_{in}$$

Truth table:

Input variables			Outputs	
X	A	B	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Karnaugh map:

	A'B'	A'B	AB	AB'
X'		1		1
X	1		1	

K-Map for Sum

	A'B'	A'B	AB	AB'
X'			1	
X		1	1	1

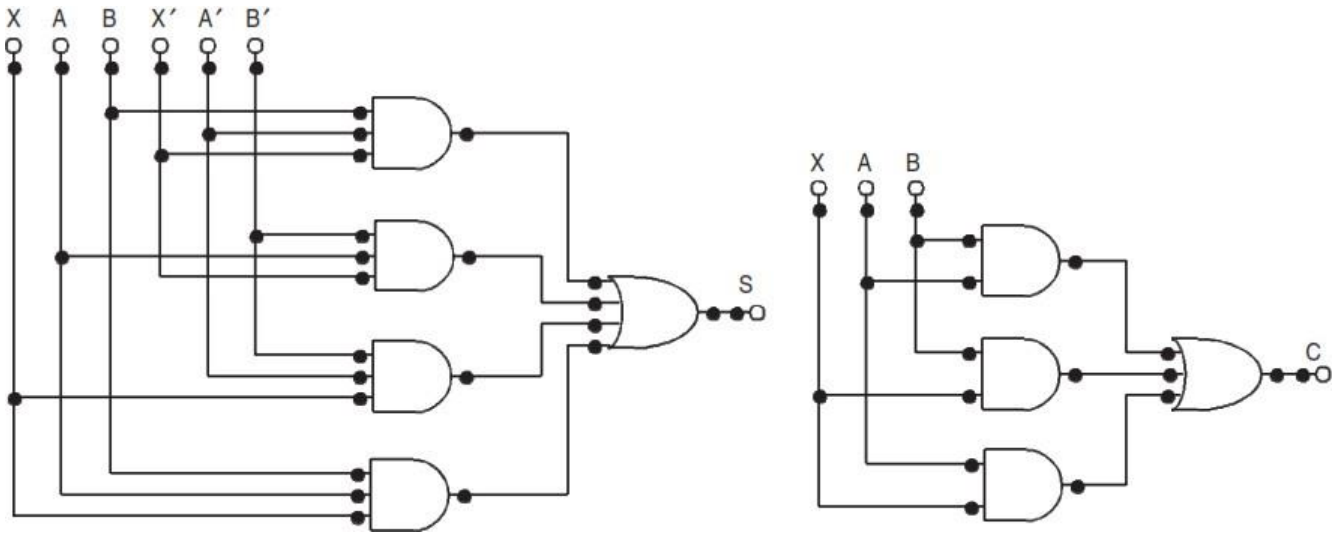
K-Map for Carry

- ❖ The simplified Boolean expressions of the outputs are

$$S = X'A'B + X'AB' + XA'B' + XAB$$

$$C = AB + BX + AX$$

Logic diagram:

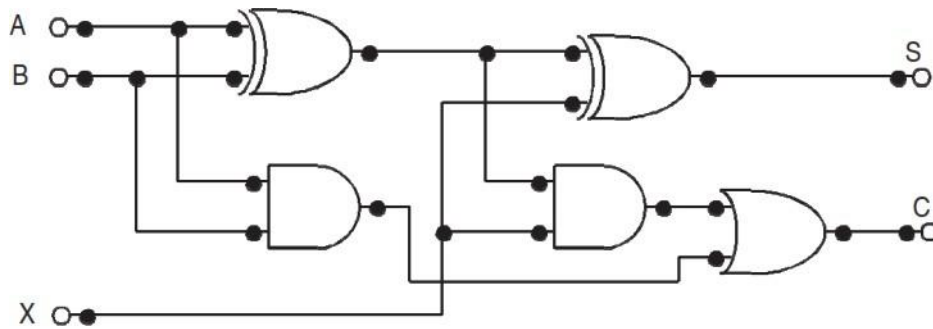


❖ The Boolean expressions of S and C are modified as follows

$$\begin{aligned}
 S &= X'A'B + X'AB' + XA'B' + XAB \\
 &= X'(A'B + AB') + X(A'B' + AB) \\
 &= X'(A \oplus B) + X(A \oplus B)' \\
 &= X \oplus A \oplus B \\
 C &= AB + BX + AX = AB + X(A + B) \\
 &= AB + X(AB + AB' + AB + A'B) \\
 &= AB + X(AB + AB' + A'B) \\
 &= AB + XAB + X(AB' + A'B) \\
 &= AB + X(A \oplus B)
 \end{aligned}$$

Full adder using Two half adder:

❖ Logic diagram according to the modified expression is shown Figure.



Half subtractor:

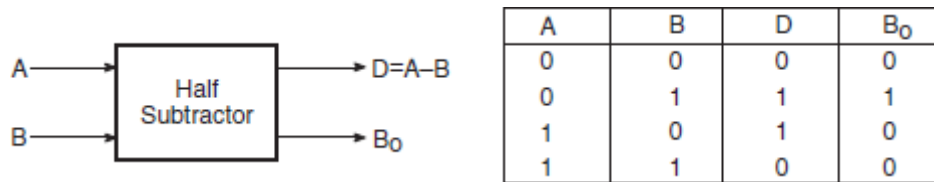
Design a half subtractor circuit.

(Nov-2009)

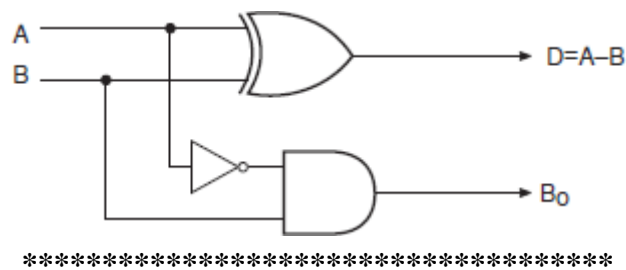
- ❖ A half-subtractor is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output.
- ❖ The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction. The Boolean expression for difference and borrow is:

$$D = \bar{A}.B + A.\bar{B}$$

$$B_0 = \bar{A}.B$$



Logic diagram:



Full subtractor:

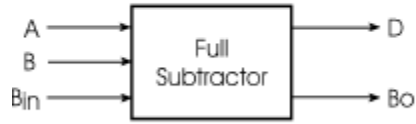
Design a full subtractor.

(Nov-2009,07)

- ❖ A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not.
- ❖ As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as Bin .
- ❖ There are two outputs, namely the DIFFERENCE output D and the BORROW output Bo. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit. The Boolean expression for difference and borrow is:

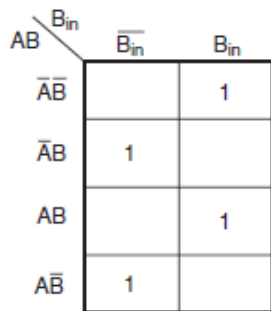
$$D = \bar{A}.\bar{B}.B_{in} + \bar{A}.B.\bar{B}_{in} + A.\bar{B}.\bar{B}_{in} + A.B.B_{in}$$

$$B_0 = \bar{A}.B + \bar{A}.B_{in} + B.B_{in}$$



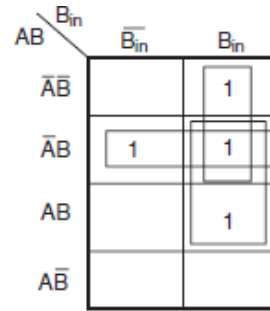
Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_o)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Map:



(a)

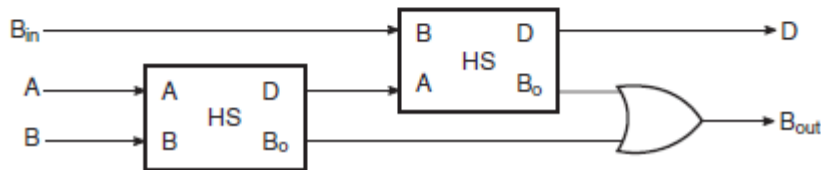
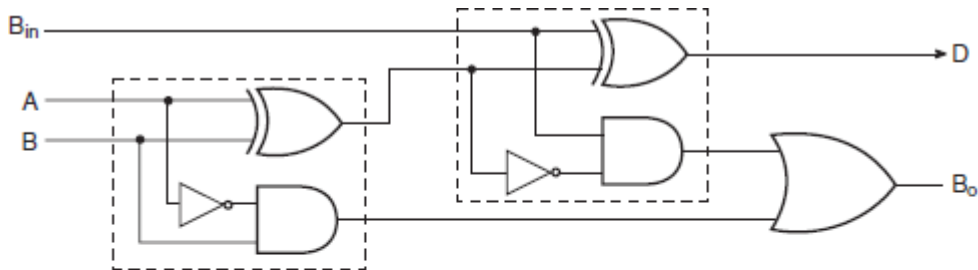
Difference



(b)

Borrow

Full subtractor using two half subtractor:



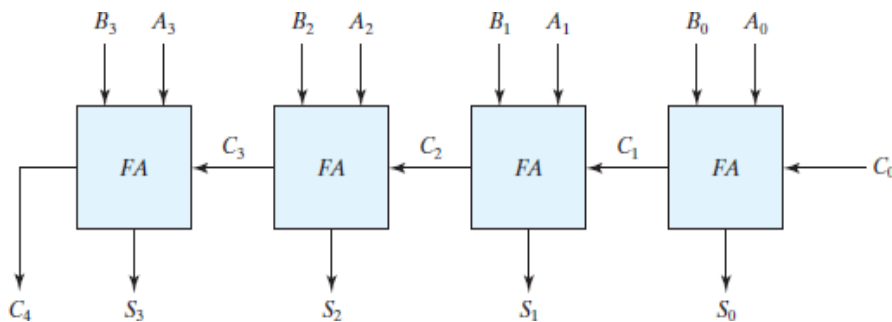
Parallel Binary Adder: (Ripple Carry Adder):

Explain about four bit adder. (or) Design of 4 bit binary adder – subtractor circuit. (Apr – 2019)

- ❖ A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- ❖ Addition of n-bit numbers requires a chain of n- full adders or a chain of one-half adder and n-1 full adders. In the former case, the input carry to the least significant position is fixed at 0.
- ❖ Figure shows the interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder.
- ❖ The carries are connected in a chain through the full adders. The input carry to the adder is C_0 , and it ripples through the full adders to the output carry C_4 . The S outputs generate the required sum bits.

Example: Consider the two binary numbers $A = 1011$ and $B = 0011$. Their sum $S = 1110$ is formed with the four-bit adder as follows:

Subscript i :	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}



- ✓ The carry output of lower order stage is connected to the carry input of the next higher order stage. Hence this type of adder is called ripple carry adder.
- ✓ In a 4-bit binary adder, where each full adder has a propagation delay of t_p ns, the output in the fourth stage will be generated only after $4t_p$ ns.
- ✓ The magnitude of such delay is prohibitive for high speed computers.
- ✓ One method of speeding up this process is look-ahead carry addition which eliminates ripple carry delay.

Complement of a number:

1's complement:

The 1's complement of a binary number is formed by changing 1 to 0 and 0 to 1.

Example:

1. The 1's complement of 1011000 is 0100111.
2. The 1's complement of 0101101 is 1010010.

2's complement:

The 2's complement of a binary number is formed by adding 1 with 1's complement of a binary number.

Example:

1. The 2's complement of 1101100 is 0010100
2. The 2's complement of 0110111 is 1001001

Subtraction using 2's complement addition:

- ✓ The subtraction of unsigned binary number can be done by means of complements.
- ✓ Subtraction of $A - B$ can be done by taking 2's complement of B and adding it to A .
- ✓ Check the resulting number. If carry present, the number is positive and remove the carry.
- ✓ If no carry present, the resulting number is negative, take the 2's complement of result and put negative sign.

Example:

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction

(a) $X - Y$ and (b) $Y - X$ by using 2's complements.

Solution:

(a) $X = 1010100$

2's complement of $Y = + 0111101$

Sum= 10010001

Discard end carry. Answer: $X - Y = 0010001$

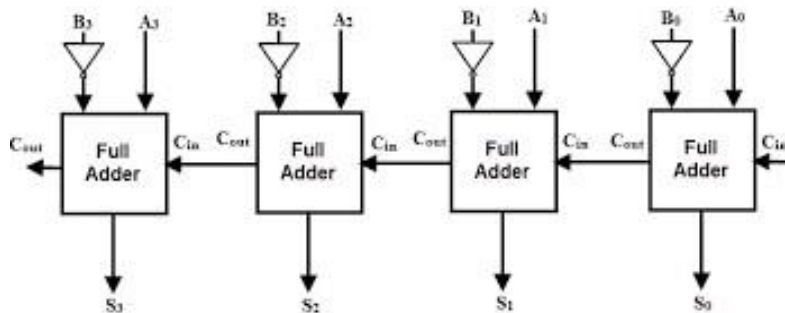
(b) $Y = 1000011$

2's complement of $X = + 0101100$

Sum= 1101111

There is no end carry. Therefore, the answer is $Y - X = -(2's \text{ complement of } 1101111) = -0010001$.

Parallel Binary Subtractor:



- ✓ The subtraction of unsigned binary numbers can be done most conveniently by means of complements. The subtraction $A - B$ can be done by taking the 2's complement of B and adding it to A . The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.
- ✓ The circuit for subtracting $A - B$ consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The input carry C_{in} must be equal to 1 when subtraction is

performed. The operation thus performed becomes A , plus the 1's complement of B , plus 1. This is equal to A plus the 2's complement of B .

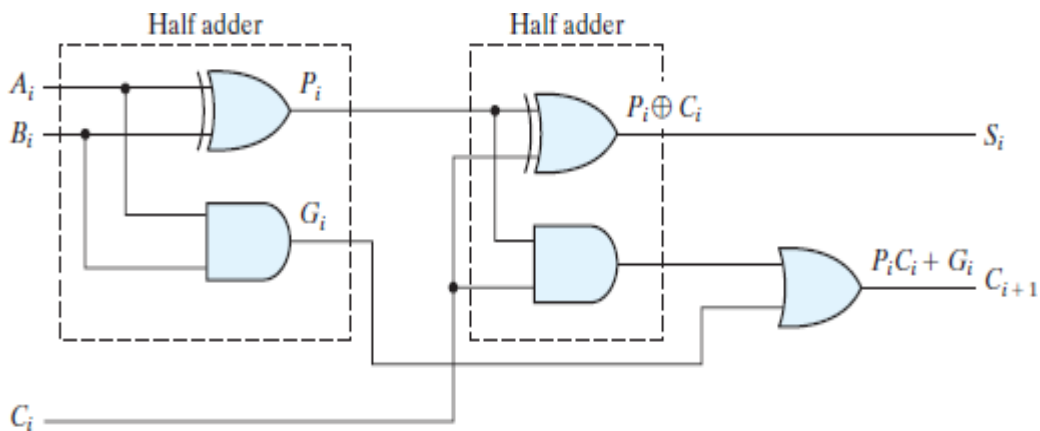
- ✓ For unsigned numbers, that gives $A-B$ if $A \geq B$ or the 2's complement of $B - A$ if $A < B$. For signed numbers, the result is $A - B$, provided that there is no overflow.

Fast adder (or) Carry Look Ahead adder:

Design a carry look ahead adder circuit.

(Nov-2010)

- ❖ The carry look ahead adder is based on the principle of looking at the lower order bits of the augend and addend to see if a higher order carry is to be generated.
- ❖ It uses two functions carry generate and carry propagate.



Consider the circuit of the full adder shown in Fig. If we define two new binary variables

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

the output sum and carry can respectively be expressed as

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

G_i is called a carry generate, and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i . P_i is called a carry propagate, because it determines whether a carry into stage i will propagate into stage $i + 1$ (i.e., whether an assertion of C_i will propagate to an assertion of C_{i+1}).

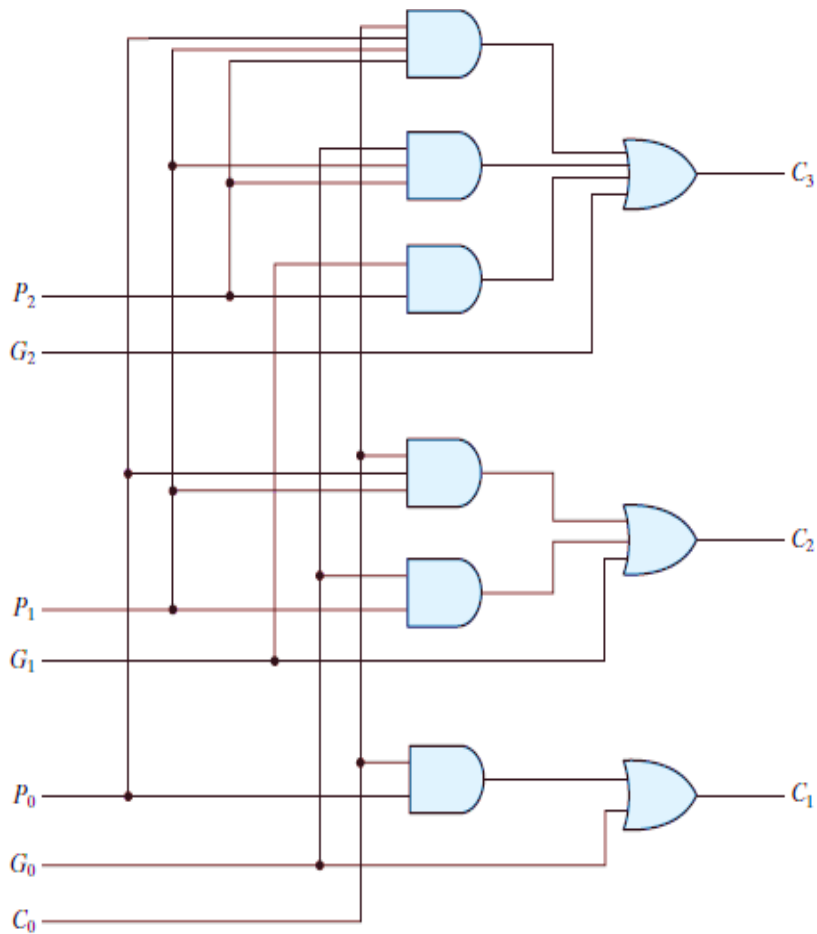
We now write the Boolean functions for the carry outputs of each stage and substitute the value of each C_i from the previous equations:

$$C_0 = \text{input carry}$$

$$C_1 = G_0 + P_0 C_0$$

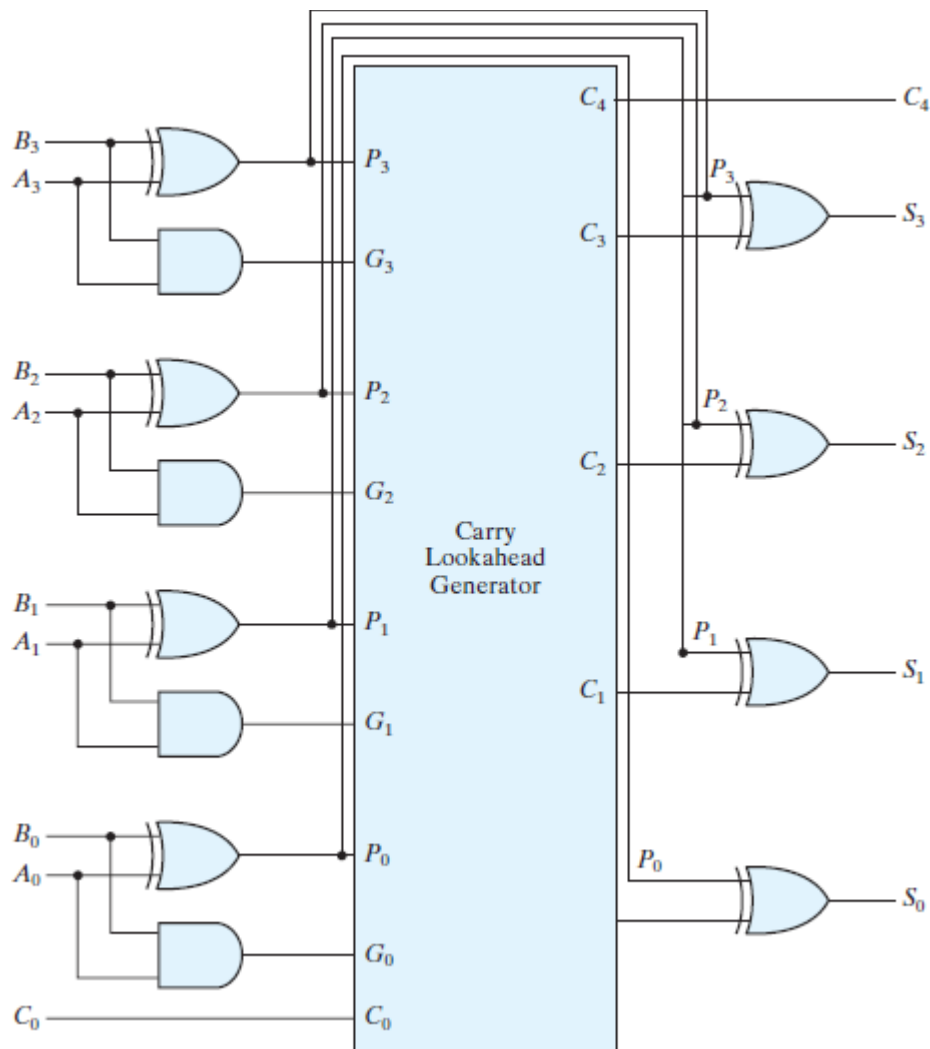
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 (G_0 + P_0 C_0) = G_1 + P_1 G_0 + P_1 P_0 C_0$$

$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 = P_2 P_1 P_0 C_0$$



Logic diagram of carry lookahead generator

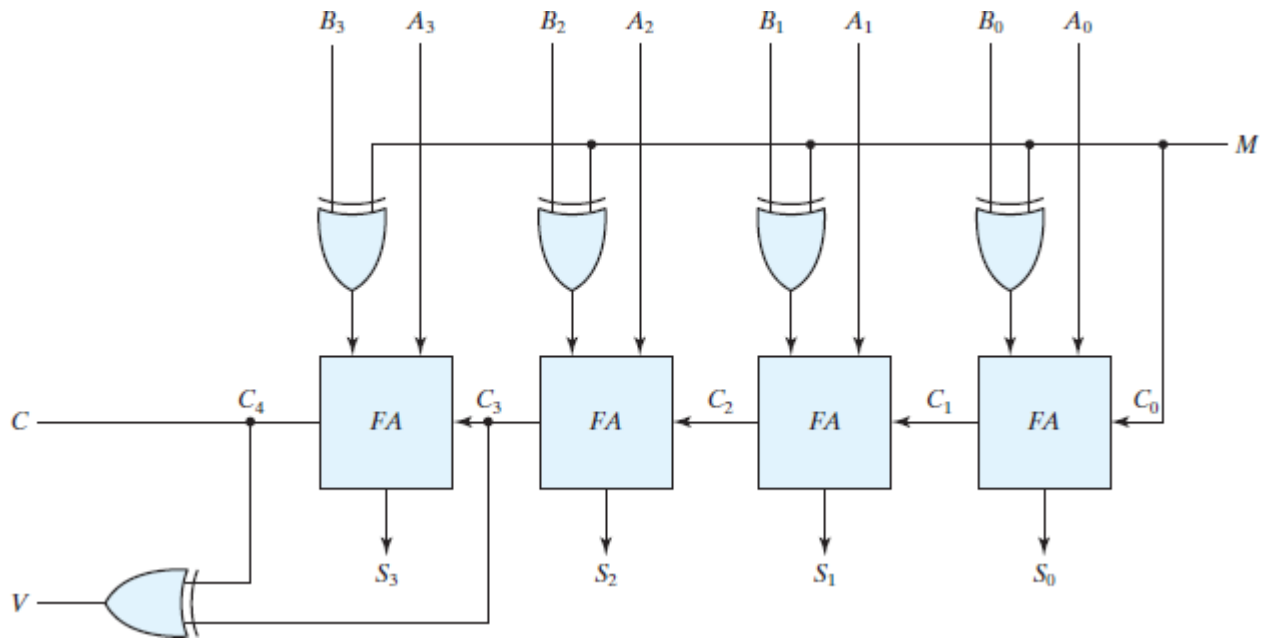
- ❖ The construction of a four-bit adder with a carry lookahead scheme is shown in Fig.
- ❖ Each sum output requires two exclusive-OR gates.
- ❖ The output of the first exclusive-OR gate generates the P_i variable, and the AND gate generates the G_i variable.
- ❖ The carries are propagated through the carry look ahead generator and applied as inputs to the second exclusive-OR gate.
- ❖ All output carries are generated after a delay through two levels of gates.
- ❖ Thus, outputs S_1 through S_3 have equal propagation delay times. The two-level circuit for the output carry C_4 is not shown. This circuit can easily be derived by the equation-substitution method.



4 bit-Parallel adder/subtractor:

Explain about binary parallel / adder subtractor. [NOV – 2019]

- ❖ The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full adder. A four-bit adder–subtractor circuit is shown in Fig.
- ❖ The mode input M controls the operation. When $M = 0$, the circuit is an adder, and when $M = 1$, the circuit becomes a subtractor.



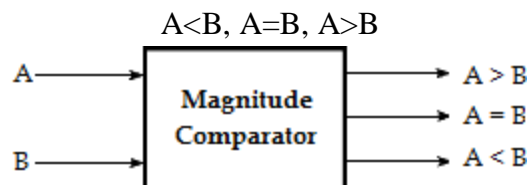
- ❖ It performs the operations of both addition and subtraction.
- ❖ It has two 4bit inputs $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$.
- ❖ The mode input M controls the operation when $M=0$ the circuit is an adder and when $M=1$ the circuits become subtractor.
- ❖ Each exclusive-OR gate receives input M and one of the inputs of B .
- ❖ When $M = 0$, we have $B \text{ xor } 0 = B$. The full adders receive the value of B , the input carry is 0, and the circuit performs A plus B . This results in sum $S_3S_2S_1S_0$ and carry C_4 .
- ❖ When $M = 1$, we have $B \text{ xor } 1 = B'$ and $C_0 = 1$. The B inputs are all complemented and a 1 is added through the input carry thus producing 2's complement of B .
- ❖ Now the data $A_3A_2A_1A_0$ will be added with 2's complement of $B_3B_2B_1B_0$ to produce the sum i.e., $A-B$ if $A \geq B$ or the 2's complement of $B-A$ if $A < B$.

Comparators

Design a 2 bit magnitude comparator.

(May 2006)

It is a combinational circuit that compares two numbers and determines their relative magnitude. The output of comparator is usually 3 binary variables indicating:



1-bit comparator: Let's begin with 1bit comparator and from the name we can easily make out that this circuit would be used to compare 1bit binary numbers.

A	B	A>B	A=B	A<B
0	0	0	1	0
1	0	1	0	0
0	1	0	0	1
1	1	0	1	0

For a 2-bit comparator we have four inputs A1 A0 and B1 B0 and three output E (is 1 if two numbers are equal) G (is 1 when A>B) and L (is 1 when A<B) If we use truth table and K-map the result is

		A>B	
		B 0	B 1
A	0	0	0
	1	1	0

Equation is $A>B = A \cdot \bar{B}$

		A<B	
		B 0	B 1
A	0	0	1
	1	0	0

Equation is $A<B = \bar{A} \cdot B$

		A=B	
		B 0	B 1
A	0	1	0
	1	0	1

The equation is $f(A=B) = \bar{A} \cdot \bar{B} + A \cdot B$
 $= A \text{ XNOR } B$

Design of 2 – bit Magnitude Comparator.

The truth table of 2-bit comparator is given in table below

Truth table:

Inputs				Outputs		
A ₃	A ₂	A ₁	A ₀	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

K-Map:

For A>B

		B ₁ B ₀			
	A ₁ A ₀	00	01	11	10
00		0	0	0	0
01		1	0	0	0
11		1	1	0	1
10		1	1	0	0

For A=B

		B ₁ B ₀			
	A ₁ A ₀	00	01	11	10
00		1	0	0	0
01		0	1	0	0
11		0	0	1	0
10		0	0	0	1

$A > B = A_0 B_1' B_0' + A_1 B_1' + A_1 A_0 B_0'$

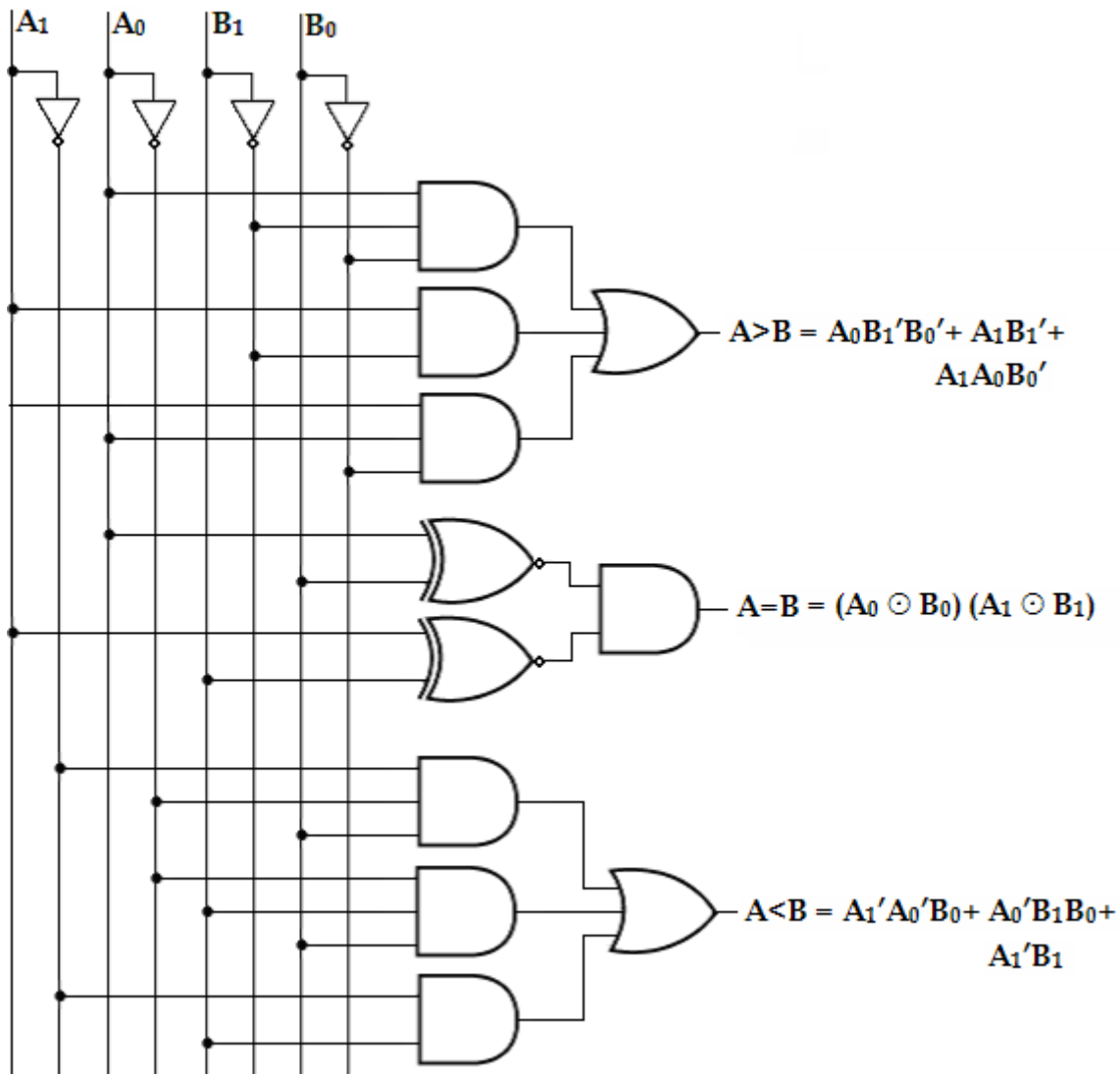
$A = B = A_1' A_0' B_1' B_0' + A_1' A_0 B_1' B_0 + A_1 A_0 B_1 B_0 + A_1 A_0' B_1 B_0'$
 $= A_1' B_1' (A_0' B_0' + A_0 B_0) + A_1 B_1 (A_0 B_0 + A_0' B_0')$
 $= (A_0 \odot B_0) (A_1 \odot B_1)$

For A<B

		B ₁ B ₀			
	A ₁ A ₀	00	01	11	10
00		0	1	1	1
01		0	0	1	1
11		0	0	0	0
10		0	0	1	0

$A < B = A_1' A_0' B_0 + A_0' B_1 B_0 + A_1' B_1$

Logic Diagram:



4 bit magnitude comparator:

Design a 4 bit magnitude comparators. (Apr – 2019)

Input

$$A = A_3 A_2 A_1 A_0$$

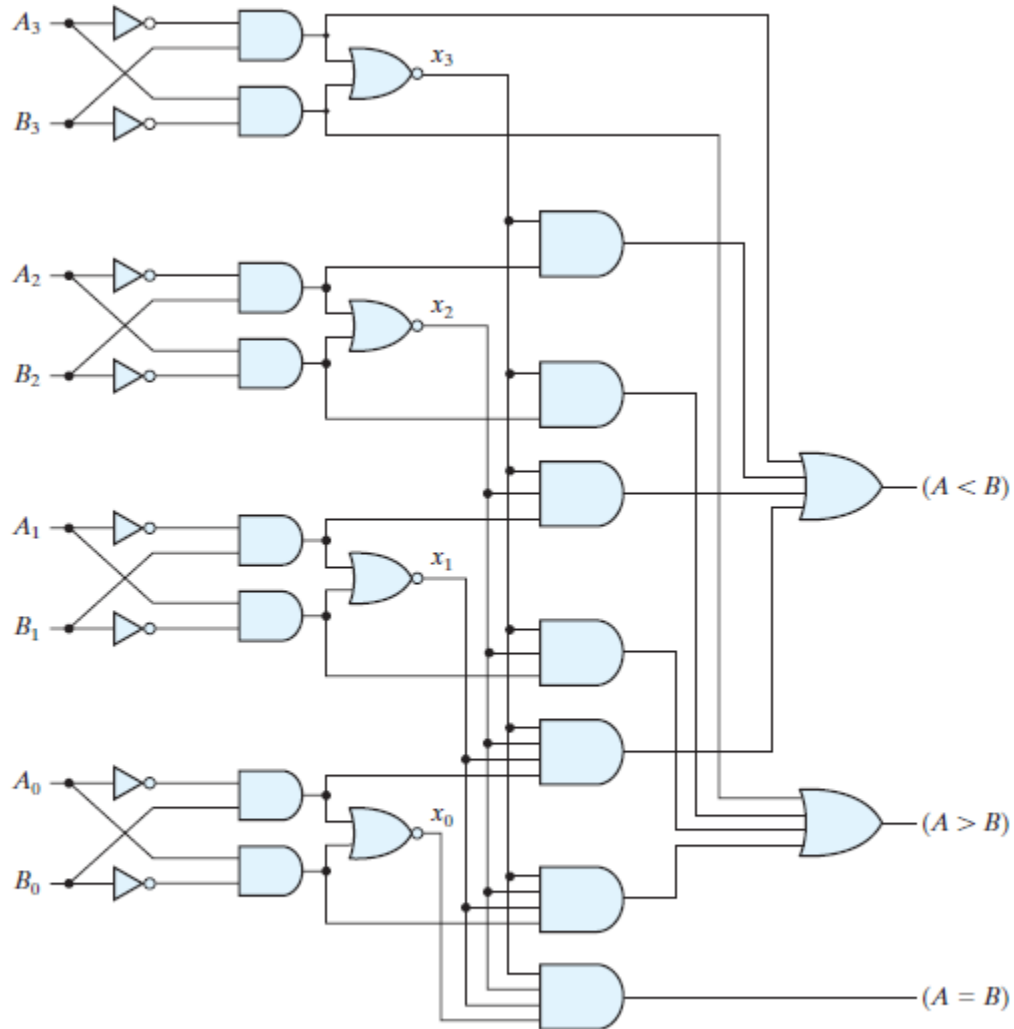
$$B = B_3 B_2 B_1 B_0$$

Function Equation

$$(A = B) = x_3x_2x_1x_0$$

$$(A > B) = A_3B'_3 + x_3A_2B'_2 + x_3x_2A_1B'_1 + x_3x_2x_1A_0B'_0$$

$$(A < B) = A'_3B_3 + x_3A'_2B_2 + x_3x_2A'_1B_1 + x_3x_2x_1A'_0B_0$$



Four-bit magnitude comparator

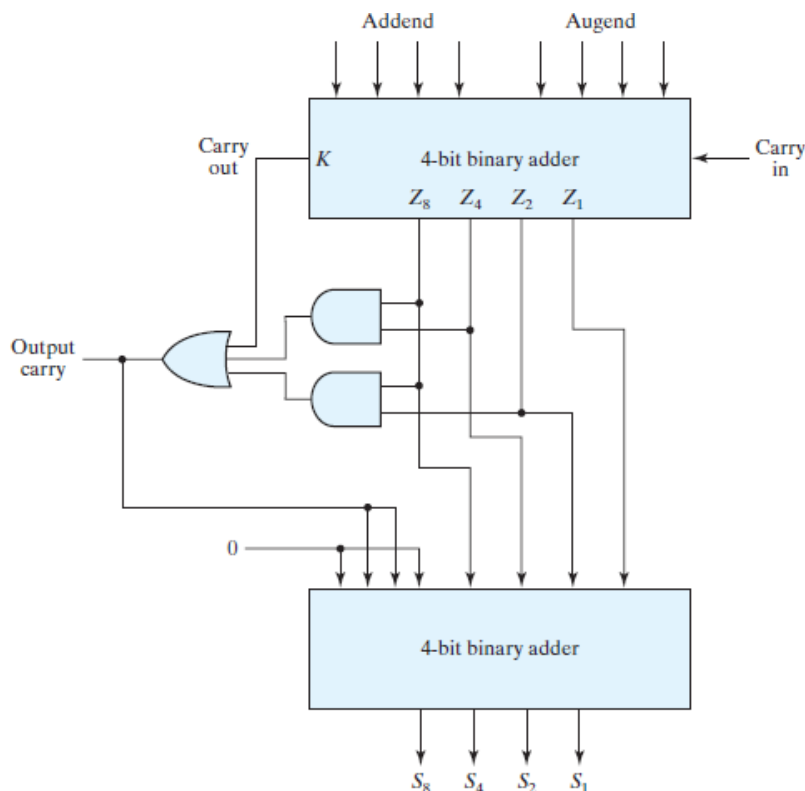
BCD Adder:

Design to perform BCD addition.(or) What is BCD adder? Design an adder to perform arithmetic addition of two decimal bits in BCD. (May -08)(Apr 2017,2018)[Nov – 2019]

- ❖ Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than $9 + 9 + 1 = 19$, the 1 in the sum being an input carry.
- ❖ Suppose we apply two BCD digits to a four-bit binary adder. The adder will form the sum in binary and produce a result that ranges from 0 through 19. These binary numbers are listed in Table and are labeled by symbols K, Z₈, Z₄, Z₂, and Z₁. K is the carry, and the subscripts under the letter Z represent the weights 8, 4, 2, and 1 that can be assigned to the four bits in the BCD code.

Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z ₈	Z ₄	Z ₂	Z ₁	C	S ₈	S ₄	S ₂	S ₁	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	0	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19



- ❖ A BCD adder that adds two BCD digits and produces a sum digit in BCD is shown in Fig. The two decimal digits, together with the input carry, are first added in the top four-bit adder to produce the binary sum.

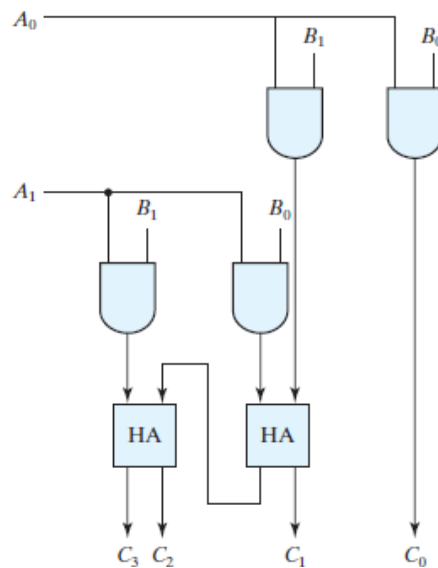
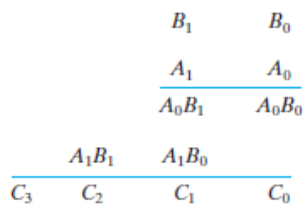
- ❖ When the output carry is equal to 0, nothing is added to the binary sum. When it is equal to 1, binary 0110 is added to the binary sum through the bottom four-bit adder.
- ❖ The condition for a correction and an output carry can be expressed by the Boolean function

$$C = K + Z_8Z_4 + Z_8Z_2$$
- ❖ The output carry generated from the bottom adder can be ignored, since it supplies information already available at the output carry terminal.
- ❖ A decimal parallel adder that adds n decimal digits needs n BCD adder stages. The output carry from one stage must be connected to the input carry of the next higher order stage.

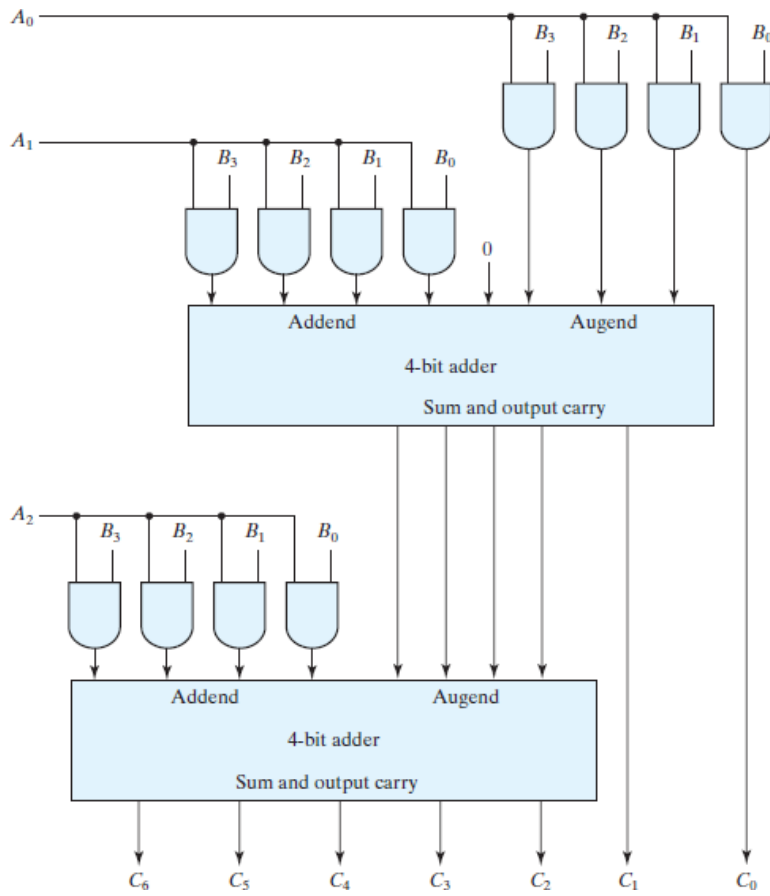
Binary Multiplier:

Explain about binary Multiplier.

- ❖ Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers. The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit. Each such multiplication forms a partial product.
- ❖ Successive partial products are shifted one position to the left. The final product is obtained from the sum of the partial products.



- ❖ A combinational circuit binary multiplier with more bits can be constructed in a similar fashion.
- ❖ A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier.
- ❖ The binary output in each level of AND gates is added with the partial product of the previous level to form a new partial product. The last level produces the product.



CODE CONVERSION

Design a binary to gray converter.

(Nov-2009)(Nov

2017)

Binary to Grayconverter

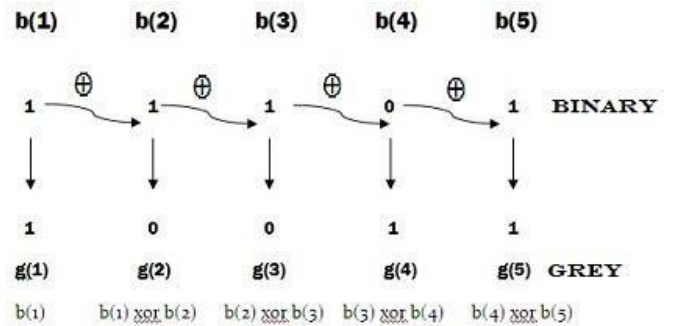
Gray code is unit distance code.

Input code: Binary [B₃ B₂ B₁ B₀]

output code: Gray [G₃ G₂ G₁ G₀]

Truth Table

B3	B2	B1	B0	G3	G2	G1	G0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0



K-MAP FORG3:

B1B0	00	01	11	10
B3B2	00	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$G3=B3$

K-MAP FORG2:

B1B0	00	01	11	10
B3B2	00	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$G2=B3'B2+B3B2'=B3 \oplus B2$

K-MAP FORG1:

K-MAP FORG0:

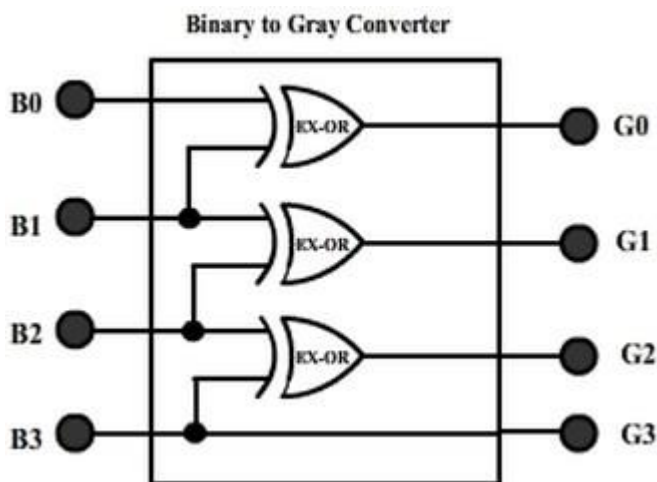
B1B0	00	01	11	10
B3B2	00	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

$$G1 = B1'B2 + B1B2' = B1 \oplus B2$$

B1B0	00	01	11	10
B3B2	00	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$$G0 = B1'B0 + B1B0' = B1 \oplus B0$$

Logic diagram:



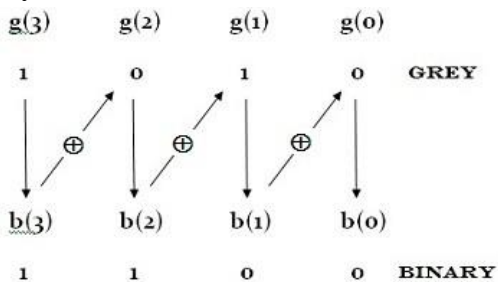
Gray to Binary converter:

Design a gray to binary converter. (OR) Design a combinational circuit that converts a four bit gray code to a four bit binary number using exclusive – OR gates. (Nov-2009) [NOV – 2019]

Gray code is unit distance code.

Input code: Gray [G₃ G₂ G₁ G₀]

output code: Binary [B₃ B₂ B₁ B₀]



i.e

$$b(3) = g(3)$$

$$b(2) = b(3) \oplus g(2)$$

$$b(1) = b(2) \oplus g(1)$$

$$b(0) = b(1) \oplus g(0)$$

Truth Table:

Gray code				Natural-binary code			
G3	G2	G1	G0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

K-Map:

For B₃

		G ₁ G ₀			
		00	01	11	10
G ₃ G ₂	00	0	0	0	0
	01	0	0	0	0
	11	1	1	1	1
	10	1	1	1	1

$$B_3 = G_3$$

For B₂

		G ₁ G ₀			
		00	01	11	10
G ₃ G ₂	00	0	0	0	0
	01	1	1	1	1
	11	0	0	0	0
	10	1	1	1	1

$$B_2 = G_3'G_2 + G_3G_2'$$

$$= G_3 \oplus G_2$$

For B₁

		G ₁ G ₀			
		00	01	11	10
G ₃ G ₂	00	0	0	1	1
	01	1	1	0	0
	11	0	0	1	1
	10	1	1	0	0

For B₀

		G ₁ G ₀			
		00	01	11	10
G ₃ G ₂	00	0	1	0	1
	01	1	0	1	0
	11	0	1	0	1
	10	1	0	1	0

From the above K-map,

$$B_3 = G_3$$

$$B_2 = G_3'G_2 + G_3G_2'$$

$$B_2 = G_3 \oplus G_2$$

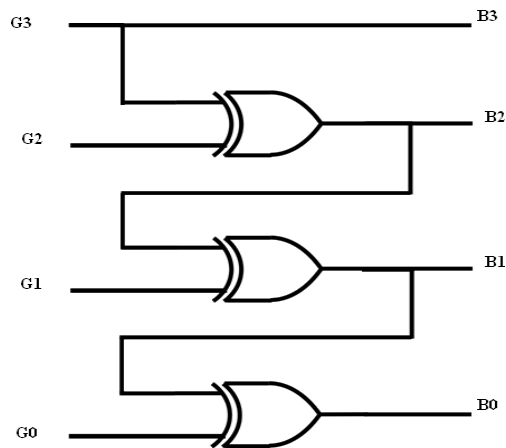
$$\begin{aligned} B_1 &= G_3'G_2'G_1 + G_3'G_2G_1' + G_3G_2G_1 + G_3G_2'G_1' \\ &= G_3' (G_2'G_1 + G_2G_1') + G_3 (G_2G_1 + G_2'G_1') \\ &= G_3' (G_2 \oplus G_1) + G_3 (G_2 \oplus G_1)' \quad [x \oplus y = x'y + xy'], [(x \oplus y)' = xy + x'y'] \end{aligned}$$

$$B_1 = G_3 \oplus G_2 \oplus G_1$$

$$\begin{aligned} B_0 &= G_3'G_2'G_1'G_0 + G_3'G_2G_1G_0' + G_3G_2G_1G_0 + G_3G_2G_1G_0' + G_3'G_2G_1'G_0' + \\ &\quad G_3G_2G_1'G_0' + G_3'G_2G_1G_0 + G_3G_2G_1G_0 \\ &= G_3'G_2' (G_1'G_0 + G_1G_0') + G_3G_2 (G_1'G_0 + G_1G_0') + G_1'G_0' (G_3'G_2 + G_3G_2') + \\ &\quad G_1G_0 (G_3'G_2 + G_3G_2') \\ &= G_3'G_2' (G_0 \oplus G_1) + G_3G_2 (G_0 \oplus G_1) + G_1'G_0' (G_2 \oplus G_3) + G_1G_0 (G_2 \oplus G_3) \\ &= G_0 \oplus G_1 (G_3'G_2' + G_3G_2) + G_2 \oplus G_3 (G_1'G_0' + G_1G_0) \\ &= (G_0 \oplus G_1) (G_2 \oplus G_3)' + (G_2 \oplus G_3) (G_0 \oplus G_1) \quad [x \oplus y = x'y + xy'] \end{aligned}$$

$$B_0 = (G_0 \oplus G_1) \oplus (G_2 \oplus G_3).$$

Logic Diagram:



BCD to Excess -3 converter:

Design a combinational circuits to convert binary coded decimal number into an excess-3 code.

- ❖ Excess-3 code is modified form of BCD code. (Nov-06,09,10, May-08,10)
- ❖ Excess -3 code is derived from BCD code by adding 3to each coded number.

Truth table:

Decimal	BCD code				Excess-3 code			
	B ₃	B ₂	B ₁	B ₀	E ₃	E ₂	E ₁	E ₀
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

K-Map:

For E₃

		B ₁ B ₀			
		00	01	11	10
B ₃ B ₂	00	0	0	0	0
	01	0	1	1	1
	11	x	x	x	x
	10	1	1	x	x

$$E_3 = B_3 + B_2 (B_0 + B_1)$$

For E₁

		B ₁ B ₀			
		00	01	11	10
B ₃ B ₂	00	1	0	1	0
	01	1	0	1	0
	11	x	x	x	x
	10	1	0	x	x

$$E_1 = B_1' B_0' + B_1 B_0$$

$$= B_1 \odot B_0$$

For E₂

		B ₁ B ₀			
		00	01	11	10
B ₃ B ₂	00	0	1	1	1
	01	1	0	0	0
	11	x	x	x	x
	10	0	1	x	x

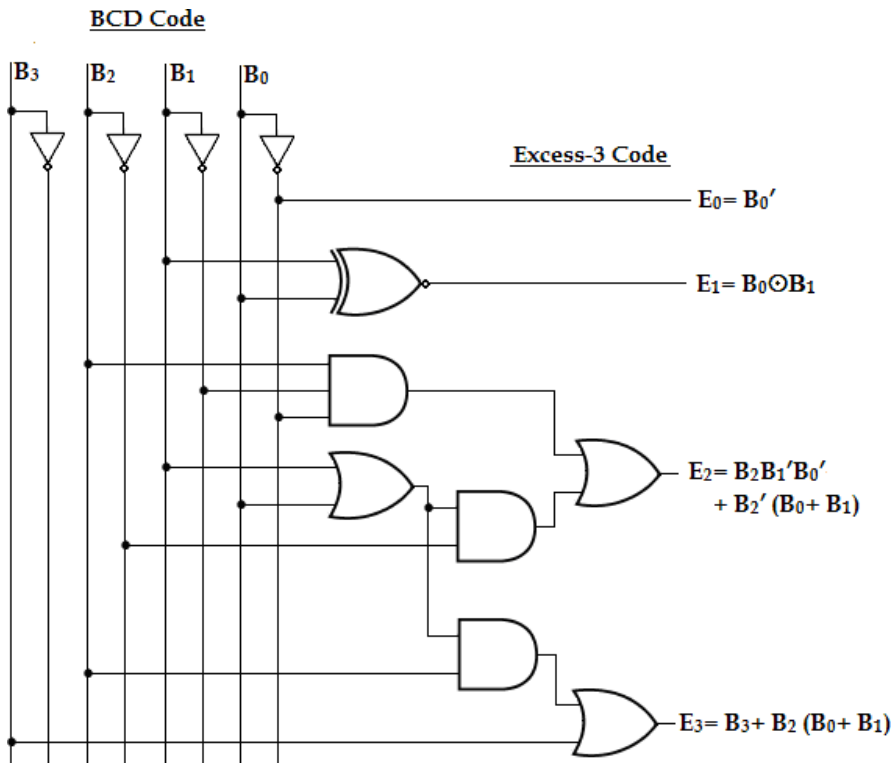
$$E_2 = B_2 B_1' B_0' + B_2' (B_0 + B_1)$$

For E₀

		B ₁ B ₀			
		00	01	11	10
B ₃ B ₂	00	1	0	0	1
	01	1	0	0	1
	11	x	x	x	x
	10	1	0	x	x

$$E_0 = B_0'$$

Logic Diagram



Excess -3 to BCD converter:

Design a combinational circuit to convert Excess-3 to BCD code.

(May 2007)

Truth table:

Decimal	Excess-3 code				BCD code			
	E ₃	E ₂	E ₁	E ₀	B ₃	B ₂	B ₁	B ₀
3	0	0	1	1	0	0	0	0
4	0	1	0	0	0	0	0	1
5	0	1	0	1	0	0	1	0
6	0	1	1	0	0	0	1	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	0	1	0	1
9	1	0	0	1	0	1	1	0
10	1	0	1	0	0	1	1	1
11	1	0	1	1	1	0	0	0
12	1	1	0	0	1	0	0	1

K-map simplification

For B_0

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	1	0	0	1
11	1	X	X	X
10	1	0	0	1

$$B_0 = \bar{E}_0$$

For B_1

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	0	1	0	1
11	0	X	X	X
10	0	1	0	1

$$B_1 = \bar{E}_1E_0 + E_1\bar{E}_0 \\ = E_1 \oplus E_0$$

For B_2

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	0	0	1	0
11	0	X	X	X
10	1	1	0	1

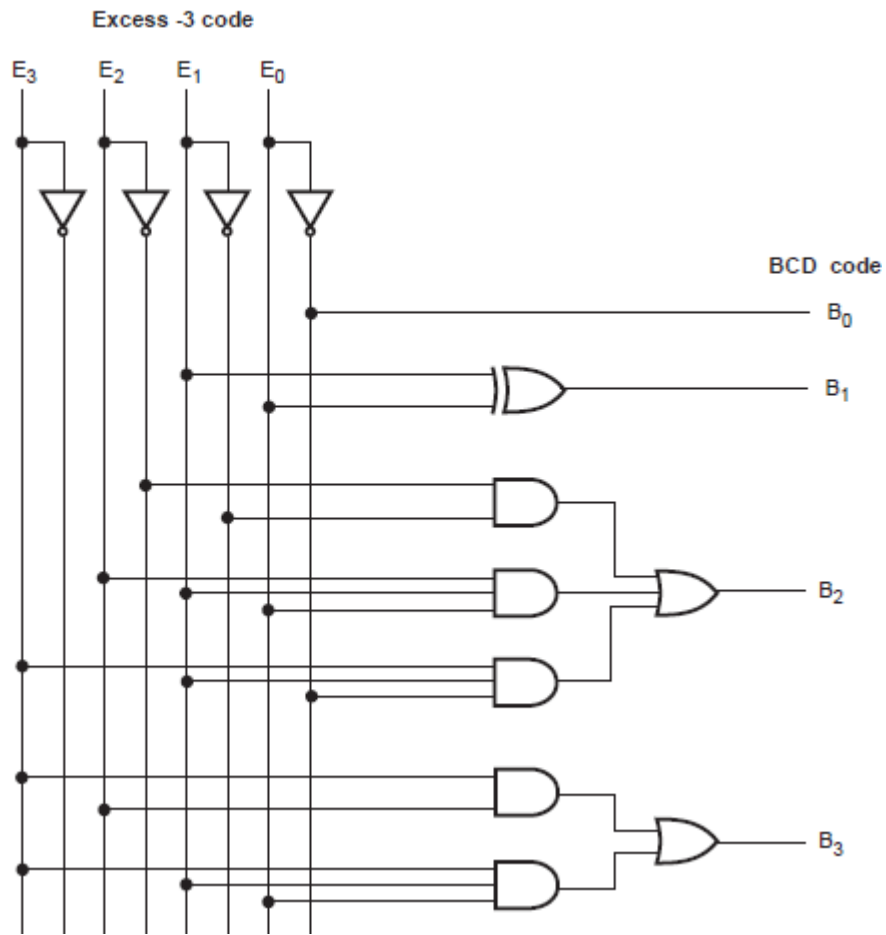
$$B_2 = \bar{E}_2\bar{E}_1 + E_2E_1E_0 + E_3E_1\bar{E}_0$$

For B_3

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	0	0	0	0
11	1	X	X	X
10	0	0	1	0

$$B_3 = E_3E_2 + E_3E_1E_0$$

Logic diagram



Design Binary to BCD converter.

Truth table:

Decimal	Binary Code				BCD Code				
	D	C	B	A	B ₄	B ₃	B ₂	B ₁	B ₀
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

K-map:

For B₀

DC \ BA	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

$B_0 = A$

For B₁

DC \ BA	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	0	0
10	0	0	0	0

$B_1 = DCB' + D'B$

For B₂

DC \ BA	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	1	1
10	0	0	0	0

$B_2 = D'C + CB$

For B₃

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

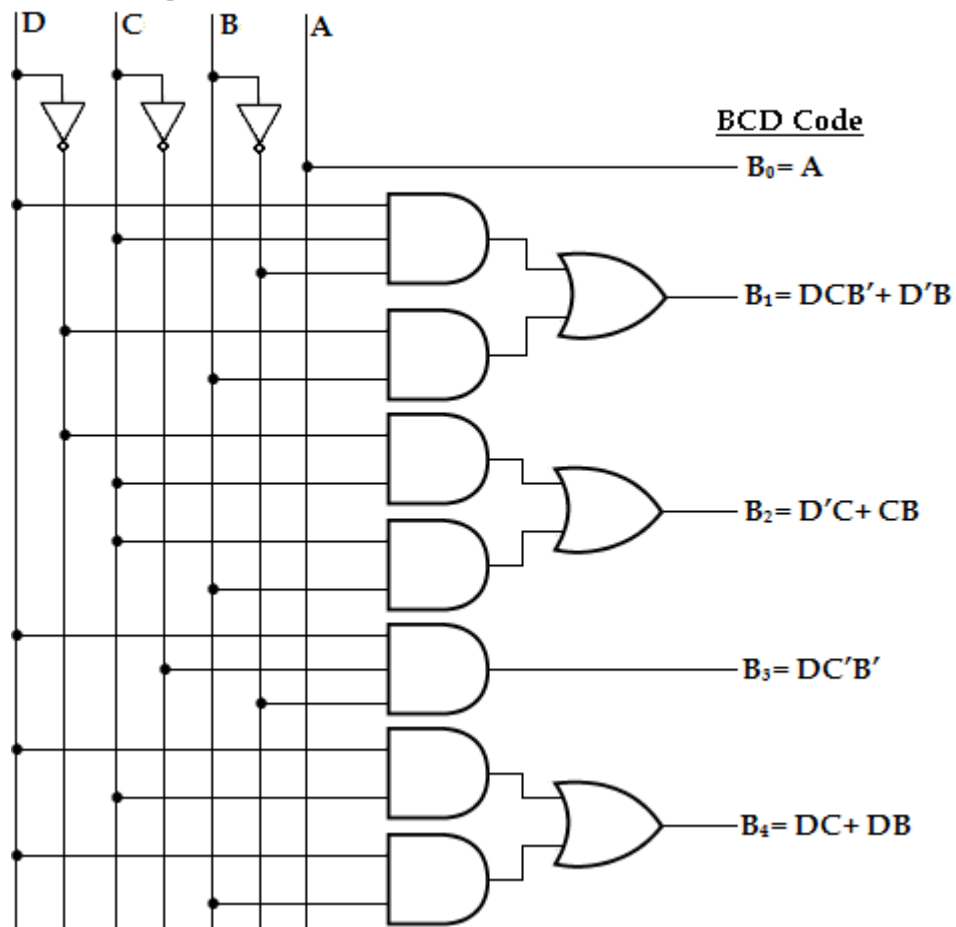
$B_3 = DC'B'$

		<u>For B_4</u>			
		DC	BA	00	01
00	00	0	0	0	0
01	01	0	0	0	0
11	11	1	1	1	1
10	10	0	0	1	1

$$B_4 = DC + DB$$

Logic diagram:

Binary Code



DECODERS AND ENCODERS

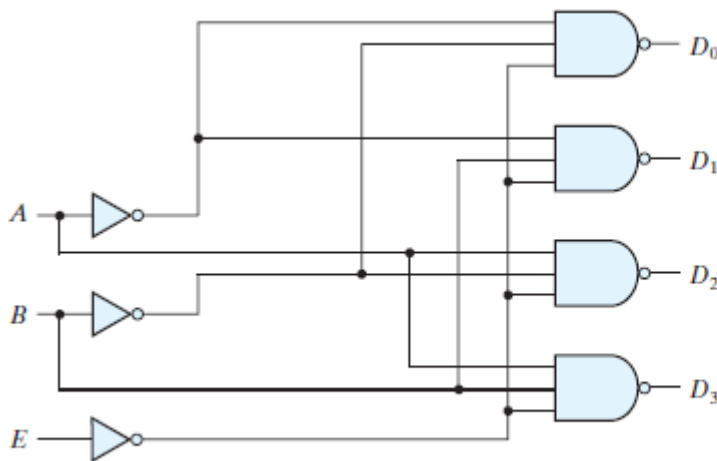
Decoder:

Explain about decoders with necessary diagrams.

(Apr 2018)(Nov 2018)

- ❖ A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n -bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.
- ❖ The purpose of a decoder is to generate the 2^n (or fewer) minterms of n input variables, shown below for two input variables.

2 to 4 decoder:



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1

(b) Truth table

3 to 8 Decoder:

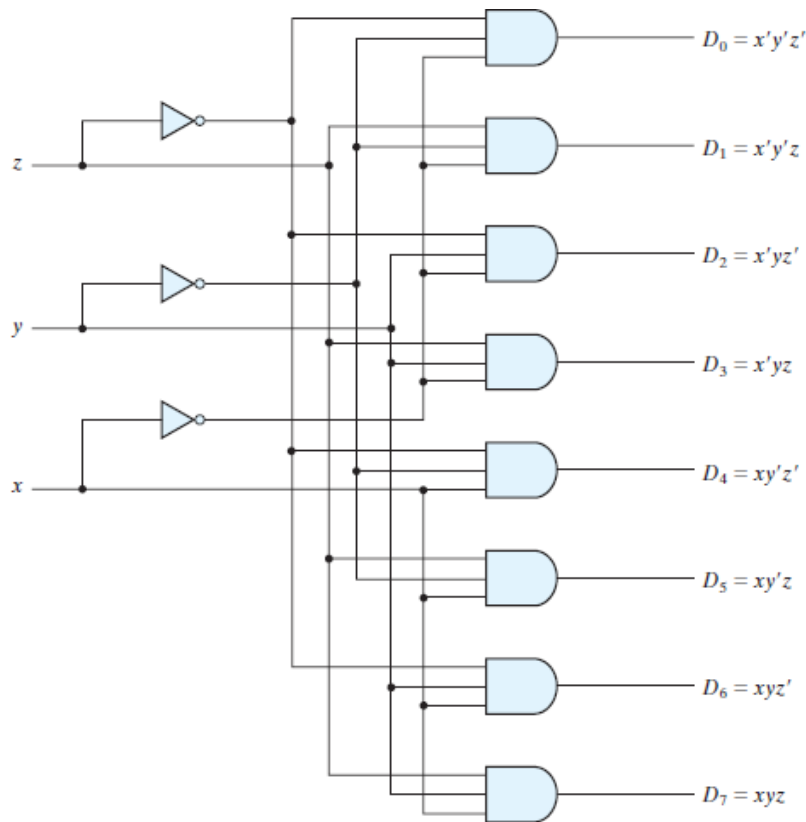
Design 3 to 8 line decoder with necessary diagram.

May -10)

Truth table:

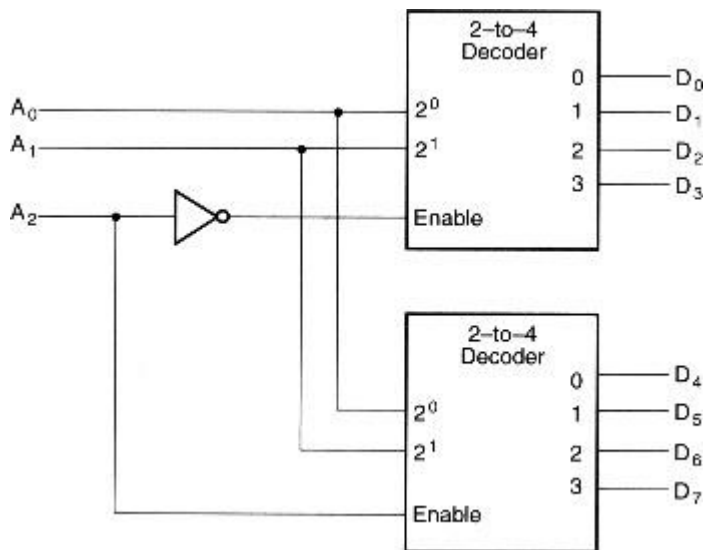
Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Logic diagram:



Design for 3 to 8 decoder with 2 to 4 decoder:

- ❖ Not that the two to four decoder design shown earlier, with its *enable* inputs can be used to build a three to eight decoder as follows.



Implementation of Boolean function using decoder:

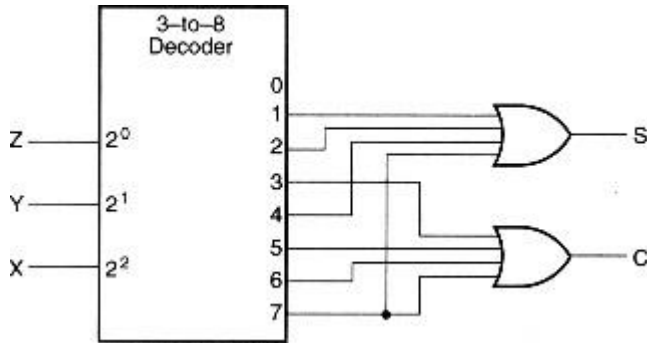
- ❖ Since the three to eight decoder provides all the minterms of three variables, the realisation of a function in terms of the sum of products can be achieved using a decoder and OR gates as follows.

Example: Implement full adder using decoder.

Sum is given by $\sum m(1, 2, 4, 7)$ while Carry is given by $\sum m(3, 5, 6, 7)$ as given by the minterms each of the OR gates are connected to.

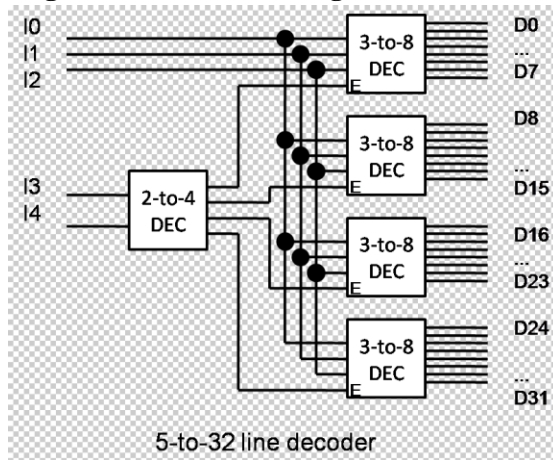
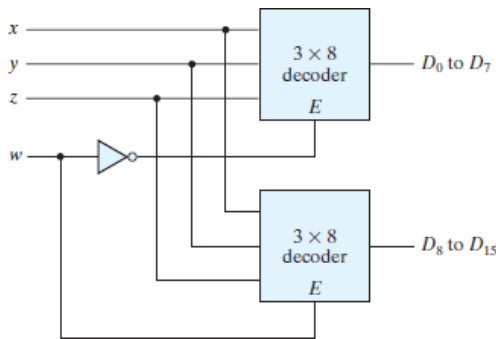
Solution :

Step 1 : Truth table



Inputs			Outputs	
A	B	C _{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

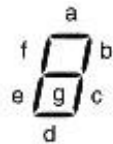
Design for 4 to 16 decoder using 3 to 8 decoder: Design 5 to 32 decoder using 3 to 8 and 2 to 4 decoder:



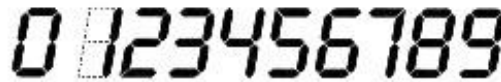
BCD to seven segment decoder

Design a BCD to seven segment code converter.

(May-06,10, Nov- 09)



(a) Segment designation

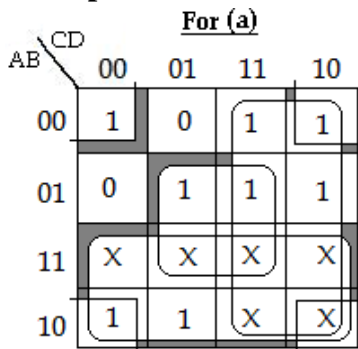


(b) Numeric designation for display

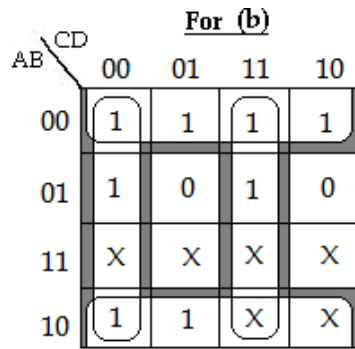
Truth table:

Digit	BCD code				7-Segment code						
	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

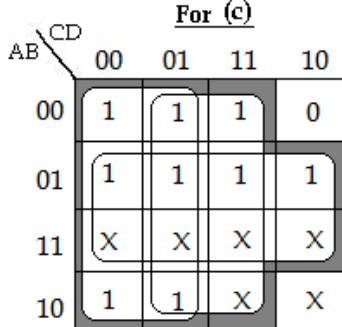
K-Map:



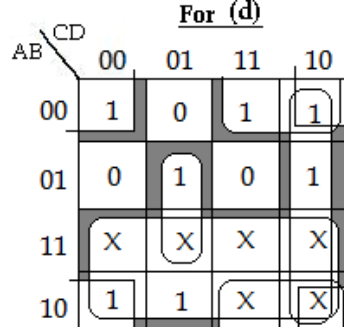
$$a = A + C + BD + B'D'$$



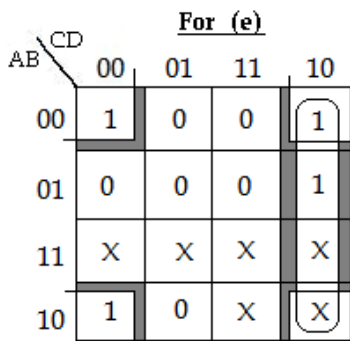
$$b = B' + C'D' + CD$$



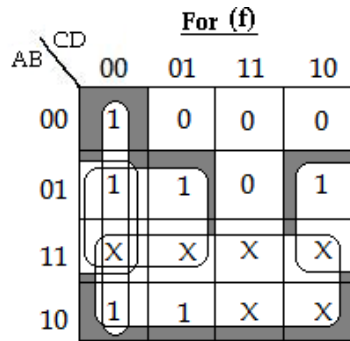
$$c = B + C' + D$$



$$d = B'D' + CD' + BC'D + B'C + A$$



$$e = B'D' + CD'$$

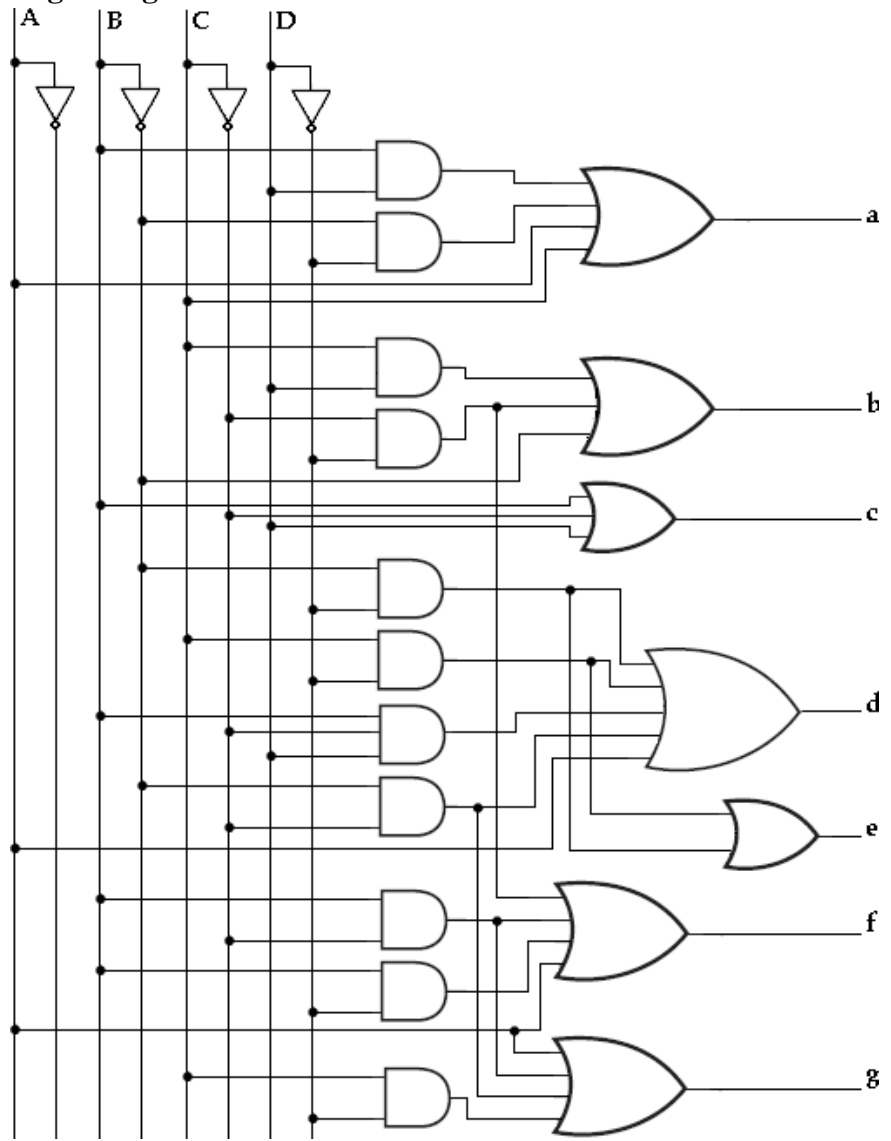


$$f = A + C'D' + BC' + BD'$$

		For (g)			
		00	01	11	10
AB	00	0	0	1	1
	01	1	1	0	1
11	X	X	X	X	
10	1	1	X	X	

$$g = A + BC' + B'C + CD'$$

Logic Diagram:



- ❖ The specification above requires that the output be zeroes (none of the segments are lighted up) when the input is not a BCD digit.
- ❖ In practical implementations, this may defer to allow representation of hexadecimal digits using the seven segments.

Encoder:

Explain about encoders. (Nov 2018)

- ❖ An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. The output lines, as an aggregate, generate the binary code corresponding to the input value.

Octal to Binary Encoder:

- ❖ The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1 when the input octal digit is 1, 3, 5, or 7.
- ❖ Output y is 1 for octal digits 2, 3, 6, or 7, and output x is 1 for digits 4, 5, 6, or 7. These conditions can be expressed by the following Boolean output functions:

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$

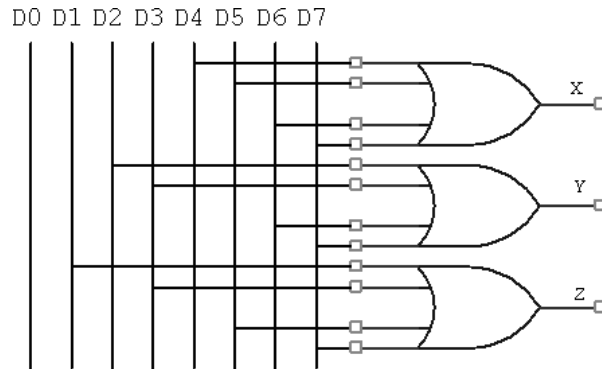
The encoder can be implemented with three OR gates.

Truth table:

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- ❖ Another ambiguity in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; but this output is the same as when D_0 is equal to 1. The discrepancy can be resolved by providing one more output to indicate whether at least one input is equal to 1.

Logic Diagram:



Priority Encoder:

Design a priority encoder with logic diagram.(or) Explain the logic diagram of a 4 – input priority encoder. (Apr – 2019)

A priority encoder is an encoder circuit that includes the priority function. The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

Truth table:

Inputs				Outputs		
D ₀	D ₁	D ₂	D ₃	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Modified Truth table:

Inputs				Outputs		
D ₀	D ₁	D ₂	D ₃	x	y	V
0	0	0	0	x	x	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
1	1	0	0			
0	0	1	0			
0	1	1	0	1	0	1
1	0	1	0			
1	1	1	0			
0	0	0	1			
0	0	1	1			
0	1	0	1			
0	1	1	1	1	1	1
1	0	0	1			
1	0	1	1			
1	1	0	1			
1	1	1	1			

K-Map:

$D_0D_1 \backslash D_2D_3$		<u>For X</u>			
		00	01	11	10
00	00	x	1	1	1
01	00	0	1	1	1
11	00	0	1	1	1
10	00	0	1	1	1

$$x = D_2 + D_3$$

$D_0D_1 \backslash D_2D_3$		<u>For Y</u>			
		00	01	11	10
00	00	x	1	1	0
01	00	1	1	1	0
11	00	1	1	1	0
10	00	0	1	1	0

$$y = D_3 + D_1D_2$$

$D_0D_1 \backslash D_2D_3$		<u>For V</u>			
		00	01	11	10
00	00	0	1	1	1
01	00	1	1	1	1
11	00	1	1	1	1
10	00	1	1	1	1

$$V = D_0 + D_1 + D_2 + D_3$$

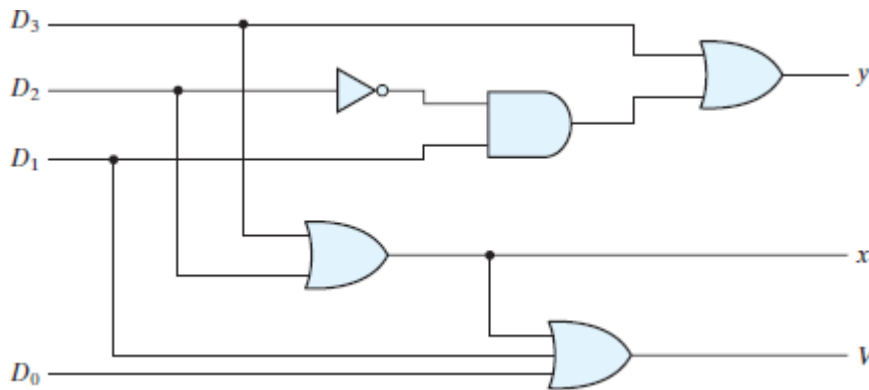
Logic Equations:

$$x = D_2 + D_3$$

$$y = D_3 + D_1 D_2'$$

$$V = D_0 + D_1 + D_2 + D_3$$

Logic diagram:



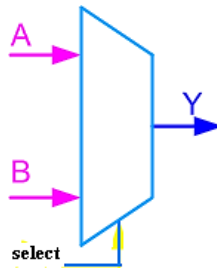
Multiplexer: (MUX)

Design a 2:1 and 4:1 multiplexer.

- ❖ A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines.
- ❖ Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

2 to 1 MUX:

A 2 to 1 line multiplexer is shown in figure below, each 2 input lines A to B is applied to one input of an AND gate. Selection lines S are decoded to select a particular AND gate. The truth table for the 2:1 mux is given in the table below.

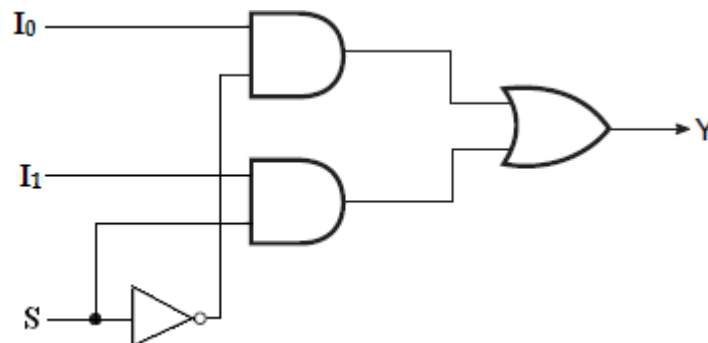


- ❖ To derive the gate level implementation of 2:1 mux we need to have truth table as shown in figure. And once we have the truth table, we can draw the K-map as shown in figure for all the cases when Y is equal to '1'.

Truth table:

S	Y
0	I_0
1	I_1

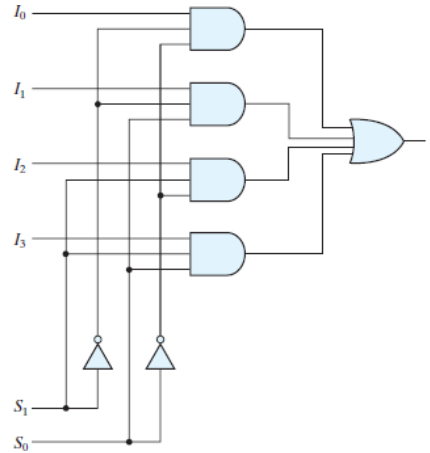
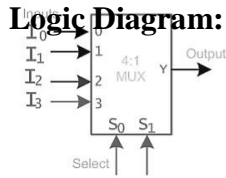
Logic Diagram:



4 to 1 MUX:

- ❖ A 4 to 1 line multiplexer is shown in figure below, each of 4 input lines I_0 to I_3 is applied to one input of an AND gate.

- ❖ Selection lines S0 and S1 are decoded to select a particular AND gate.
- ❖ The truth table for the 4:1 mux is given in the table below.



Truth Table:

SELECT INPUT		OUTPUT
S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Problems :

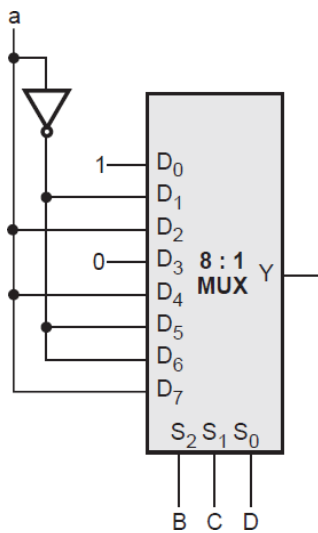
Example: Implement the Boolean expression using MUX

$$F(A,B,C,D) = \sum m(0,1,5,6,8,10,12,15)$$

(Apr 2017, Nov 2017)

Solution : Implementation table :

	D ₀	D ₁	D ₂	D ₃	D ₄	D ₅	D ₆	D ₇
\bar{a}	0	1	2	3	4	5	6	7
a	8	9	10	11	12	13	14	15
	1	\bar{a}	a	0	a	\bar{a}	\bar{a}	a



Example: Implement the boolean function using Multiplexer. [NOV – 2019]

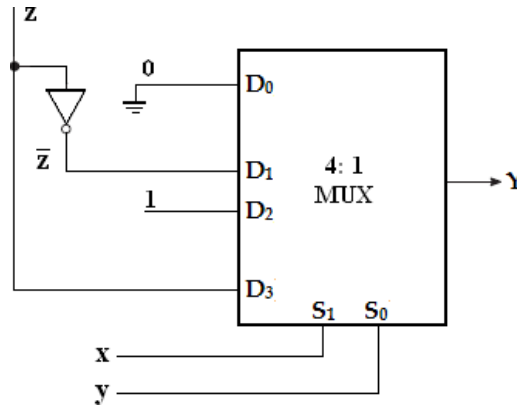
$$F(x, y, z) = \sum m(1, 2, 6, 7)$$

Solution:

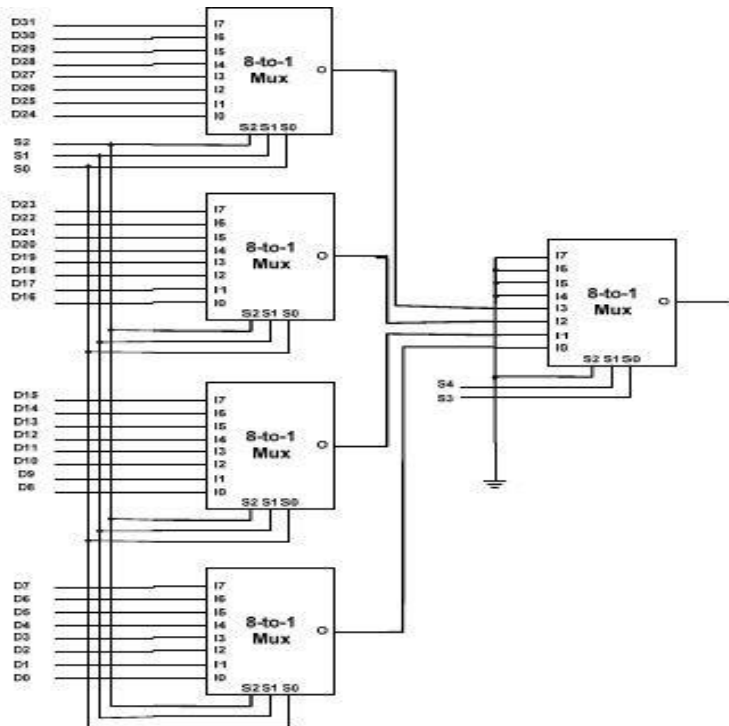
Implementation table:

	D ₀	D ₁	D ₂	D ₃
\bar{z}	0	1	2	3
z	4	5	6	7
	0	\bar{z}	1	z

Multiplexer Implementation:



Example: 32:1 Multiplexer using 8:1 Mux (Nov 2018) (Apr – 2019)

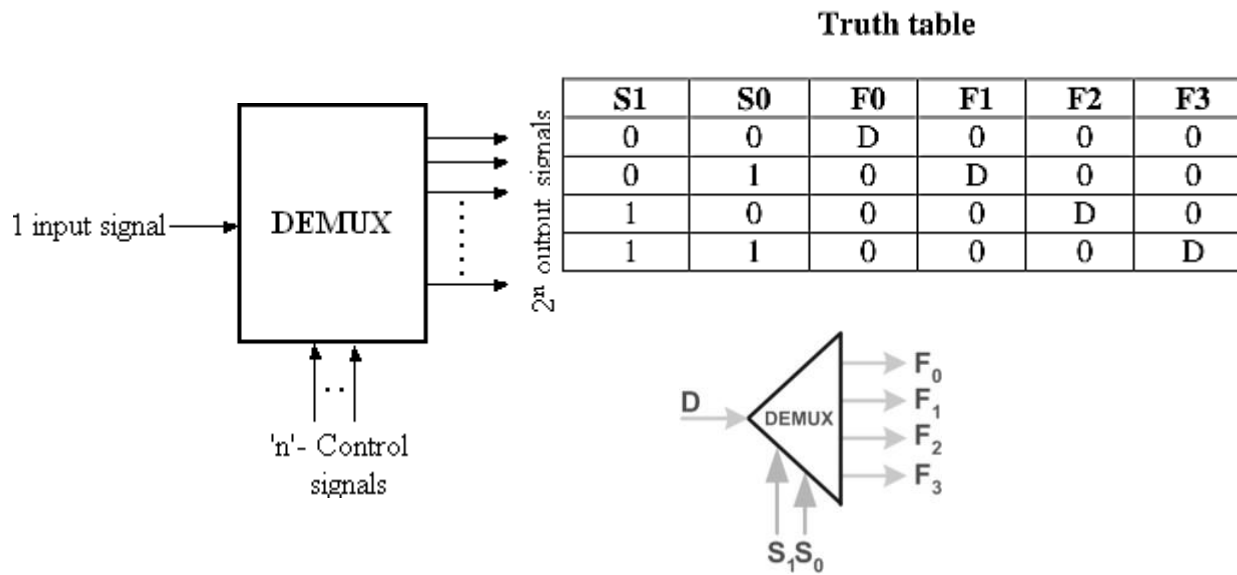


DEMULTIPLEXERS:

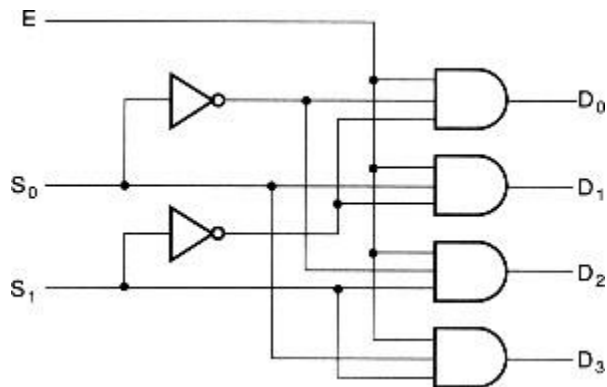
Explain about demultiplexers.

- ❖ The de-multiplexer performs the inverse function of a multiplexer, that is it receives information on one line and transmits its onto one of 2n possible output lines.

❖ The selection is by n input select lines. Example: 1-to-4 De-multiplexer



Logic Diagram:



Truth Table:

INPUT				OUTPUT			
E	D	S0	S1	Y0	Y1	Y2	Y3
1	1	0	0	1	0	0	0
1	1	0	1	0	1	0	0
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

Example:

1. Implement full adder using De-multiplexer.

Solution :

Step 1 : Truth table

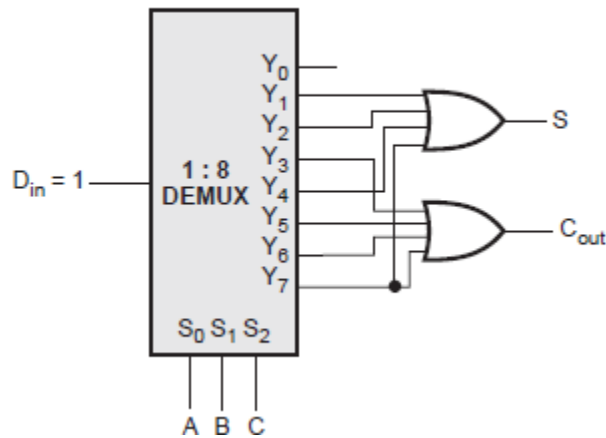
Inputs			Outputs	
A	B	C _{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Step 2 : For full adder

$$\text{Carry} = C_{\text{out}} = \sum m (3, 5, 6, 7)$$

and $\text{Sum} = S = \sum m (1, 2, 4, 7)$

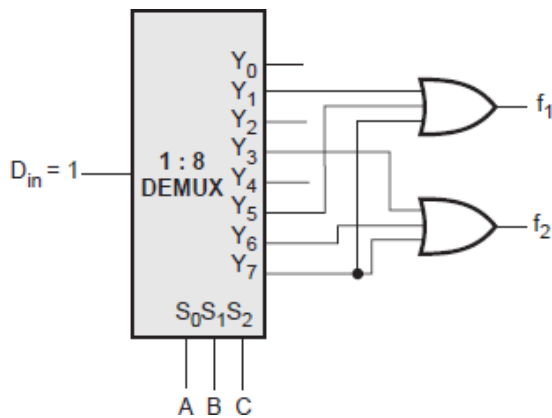
Step 3 : When D_{in} = 1, the demultiplexer gives minterms at the output.



2. Implement the following functions using de-multiplexer.

$$f_1(A,B,C) = \sum m(1,5,7), f_2(A,B,C) = \sum m(3,6,7)$$

Solution:



Parity Checker / Generator:

- A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The message, including the parity bit, is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted.
- The circuit that generates the parity bit in the transmitter is called a *parity generator*. The circuit that checks the parity in the receiver is called a *parity checker*.
- In even parity system, the parity bit is '0' if there are even number of 1s in the data and the parity bit is '1' if there are odd number of 1s in the data.
- In odd parity system, the parity bit is '1' if there are even number of 1s in the data and the parity bit is '0' if there are odd number of 1s in the data.

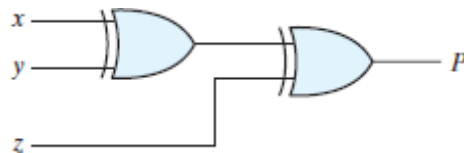
3-bit Even Parity generator:

Truth Table:

Three-Bit Message			Parity Bit
<i>x</i>	<i>y</i>	<i>z</i>	<i>P</i>
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

$$P = x \oplus y \oplus z$$

Logic Diagram:



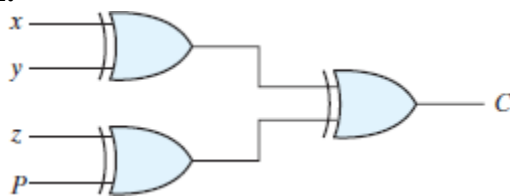
4-bit Even parity checker:

Truth Table:

Four Bits Received				Parity Error Check
<i>x</i>	<i>y</i>	<i>z</i>	<i>P</i>	<i>C</i>
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

$$C = x \oplus y \oplus z \oplus P$$

Logic Diagram:



INTRODUCTION TO HDL

- ❖ In electronics, a **hardware description language or HDL** is any language from a class of computer languages and/or programming languages for formal description of digital logic and electronic circuits.
- ❖ HDLs are used to write executable specifications of some piece of hardware.
- ❖ A simulation program, designed to implement the underlying semantics of the language statements, coupled with simulating the progress of time, provides the hardware designer with the ability to model a piece of hardware before it is created physically.
- ❖ *Logic synthesis* is the process of deriving a list of components and their interconnection (called net list) from the model of a digital system.
- ❖ *Logic Simulation* is the representation of the structure and behavior of a digital logic synthesis through the use of a computer.
- ❖ The standard HDLs that supported by IEEE.
 - ✓ VHDL (very High Speed Integrated Circuit HDL)
 - ✓ Verilog HDL

HDL MODELS OF COMBINATIONAL CIRCUITS

The Verilog HDL model of a combinational circuit can be described in any one of the following modeling styles,

- ✓ Gate level modeling-using instantiations of predefined and user defined primitive gates.
- ✓ Dataflow modeling using continuous assignment with the keyword **assign**.
- ✓ Behavioral modeling using procedural assignment statements with the keyword **always**.

Gate level modeling

In this type, a circuit is specified by its logic gates and their interconnections. Gate level modeling provides a textual description of a schematic diagram. The verilog HDL includes 12 basic gates as predefined primitives. They are and, nand, or, nor, xor, xnor, not & buf.

HDL

```
// Gate-level description of two-to-four-line decoder
// Refer to Fig. 4.19 with symbol E replaced by enable, for clarity.

module decoder_2x4_gates (D, A, B, enable);
    output      [0: 3]    D;
    input       A, B;
    input       enable;
    wire       A_not, B_not, enable_not;

    not
        G1 (A_not, A),
        G2 (B_not, B),
        G3 (enable_not, enable);
    nand
        G4 (D[0], A_not, B_not, enable_not),
        G5 (D[1], A_not, B, enable_not),
        G6 (D[2], A, B_not, enable_not),
        G7 (D[3], A, B, enable_not);
endmodule
```

Data flow modeling

Data flow modeling of combinational logic uses a number of operators that act on operands to produce desired results. Verilog HDL provides about 30 different operators. Data flow modeling uses continuous assignments and the keyword **assign**. A continuous assignment is a statement that assigns a value to a net. The data type family **net** is used to represent a physical connection between circuit elements.

HDL for 2-to-4 line decoder

Symbol	Operation	Verilog Code
+	binary addition	<pre> module decoder_2x4_df (output [0: 3] D, input A, B, enable); assign D[0] = ~(~A & ~B & ~enable), D[1] = ~(~A & B & ~enable), D[2] = ~(A & ~B & ~enable), D[3] = ~(A & B & ~enable); endmodule </pre>
-	binary subtraction	
&	bitwise AND	
	bitwise OR	
^	bitwise XOR	
~	bitwise NOT	
==	equality	
>	greater than	
<	less than	
{ }	concatenation	
?:	conditional	

Behavioral modeling

- ❖ Behavioral modeling represents digital circuits at a functional and algorithmic level. It is used mostly to describe sequential circuits, but can also be used to describe combinational circuits.
- ❖ Behavioral descriptions use the keyword **always**, followed by an optional event control expression and a list of procedural assignment statements.

```

// Behavioral description of two-to-one-line multiplexer

module mux_2x1_beh (m_out, A, B, select);
  output      m_out;
  input       A, B, select;
  reg         m_out;

  always      @(A or B or select)
    if (select == 1) m_out = A;
    else m_out = B;
endmodule

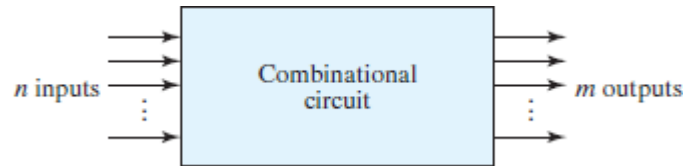
```

UNIT II COMBINATIONAL LOGIC

TWO MARK QUESTIONS & ANSWERS

1) Define combinational logic. (May 2008, 2016)

A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs. A combinational circuit performs an operation that can be specified logically by a set of Boolean functions.



2) What are sequential circuits?

Sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements. Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states.

3) Write the design procedure for combinational circuits?

The procedure involves the following steps:

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design (manually or by simulation).

4) What is Half adder?

A half-adder is an arithmetic circuit block that can be used to add two bits and produce two outputs SUM and CARRY.

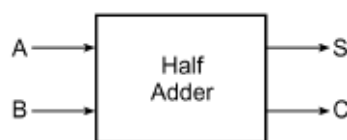
The Boolean expressions for the SUM and CARRY outputs are given by the equations

$$\text{SUM } S = A.\bar{B} + \bar{A}.B$$

$$\text{CARRY } C = A.B$$

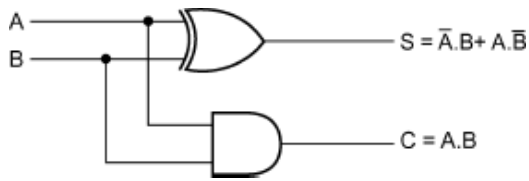
Truth Table:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

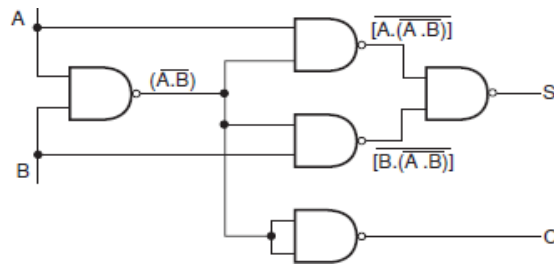


5) Draw the logic diagram of half adder using NAND gate. (May 2006,13)

Logic Diagram:



Half adder using NAND gate:



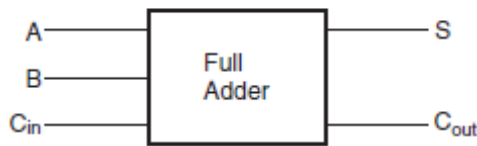
6) What is Full adder? Draw the truth table of full adder. (Apr 2018)

A Full-adder is an arithmetic circuit block that can be used to add three bits and produce two outputs SUM and CARRY.

The Boolean expressions for the SUM and CARRY outputs are given by the equations

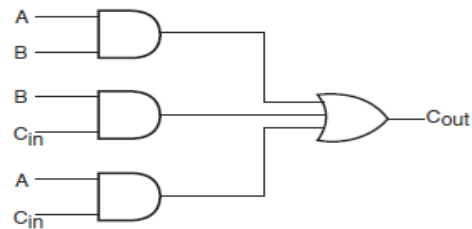
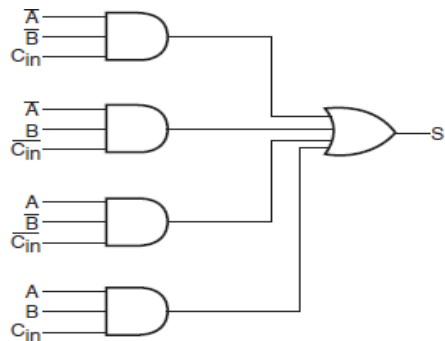
$$S = \bar{A}.\bar{B}.C_{in} + \bar{A}.B.\bar{C}_{in} + A.\bar{B}.\bar{C}_{in} + A.B.C_{in}$$

$$C_{out} = B.C_{in} + A.B + A.C_{in}$$



A	B	C _{in}	SUM (S)	C _{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

7) Draw the Logic diagram of full adder.

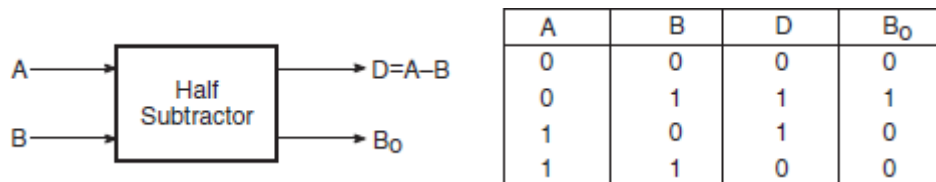


8) What is Half subtractor? (May 2005)

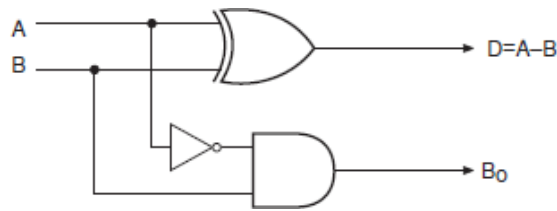
A half-subtractor is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a '1' has been borrowed to perform the subtraction. The Boolean expression for difference and borrow is:

$$D = \bar{A}.B + A.\bar{B}$$

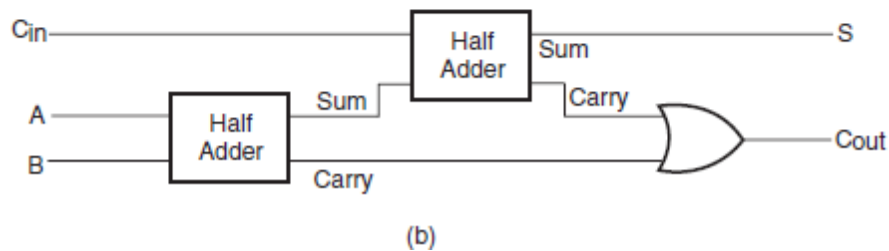
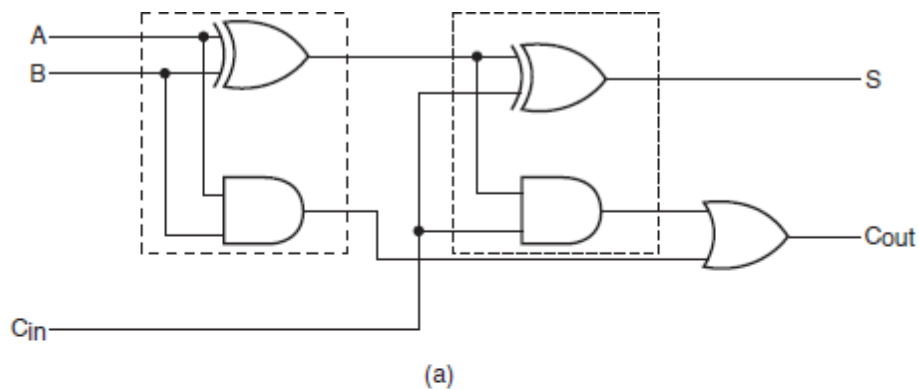
$$B_0 = \bar{A}.B$$



Logic diagram:

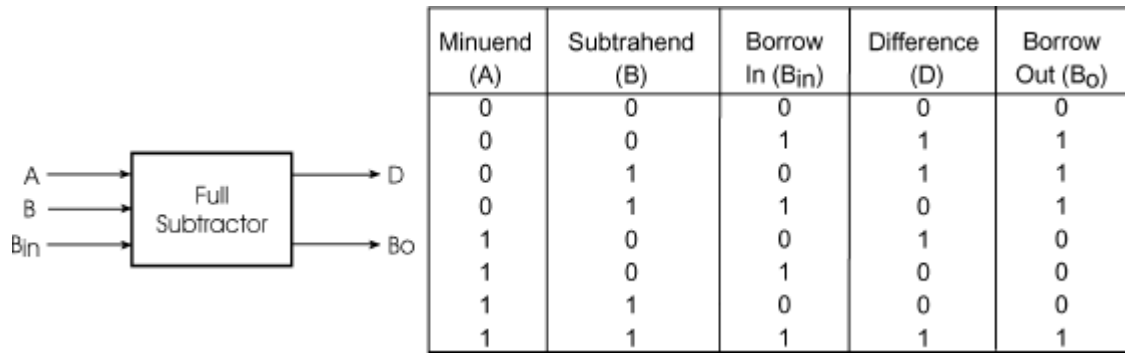


9) Draw Full adder using Two half adder. (Apr – 2019)

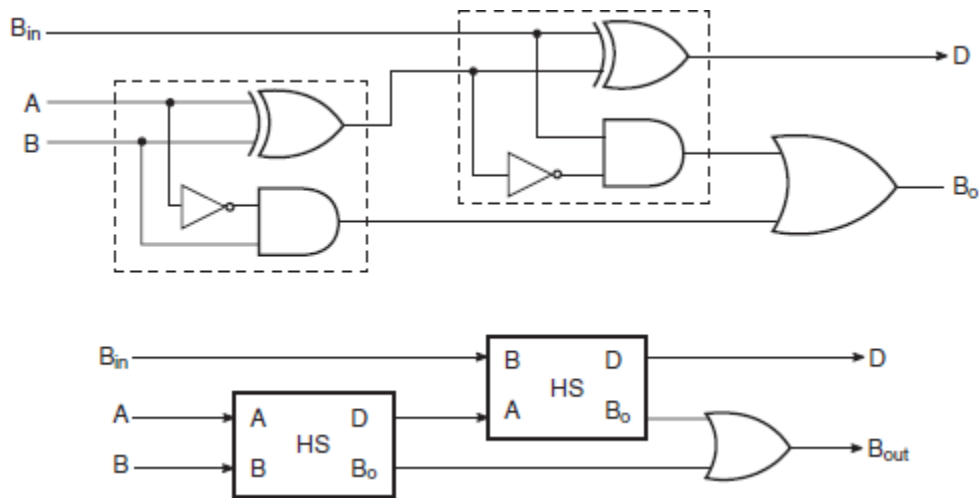


10) What is Full subtractor? Write the truth table of full subtractor. (Nov 2017)

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a '1' has already been borrowed by the previous adjacent lower minuend bit or not. As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as Bin . There are two outputs, namely the DIFFERENCE output D and the BORROW output Bo. The BORROW output bit tells whether the minuend bit needs to borrow a '1' from the next possible higher minuend bit. The Boolean expression for difference and borrow is:



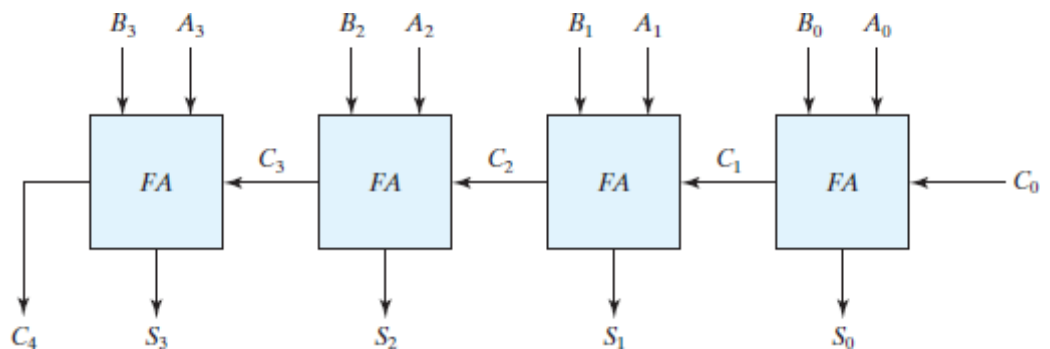
11) Draw Full subtractor using two half subtractor.



12) What is Parallel Binary Adder (Ripple Carry Adder)?

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.

13) Draw the logic diagram for four bit binary parallel adder.



14) What is 1's complement of a number?

The 1's complement of a binary number is formed by changing 1 to 0 and 0 to 1.

Example:

1. The 1's complement of 1011000 is 0100111.
2. The 1's complement of 0101101 is 1010010.

15) What is 2's complement of a number?

The 2's complement of a binary number is formed by adding 1 with 1's complement of a binary number.

Example:

- 1) The 2's complement of 1101100 is 0010100
- 2) The 2's complement of 0110111 is 1001001

16) How Subtraction of binary numbers perform using 2's complement addition?

- ✓ The subtraction of unsigned binary number can be done by means of complements.
- ✓ Subtraction of $A-B$ can be done by taking 2's complement of B and adding it to A .
- ✓ Check the resulting number. If carry present, the number is positive and remove the carry.
- ✓ If no carry present, the resulting number is negative, take the 2's complement of result and put negative sign.

17) Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction

(a) $X - Y$ and (b) $Y - X$ by using 2's complements.

Solution:

(c) $X = 1010100$

2's complement of $Y = + 0111101$

Sum= 10010001

Discard end carry. *Answer:* $X - Y = 0010001$

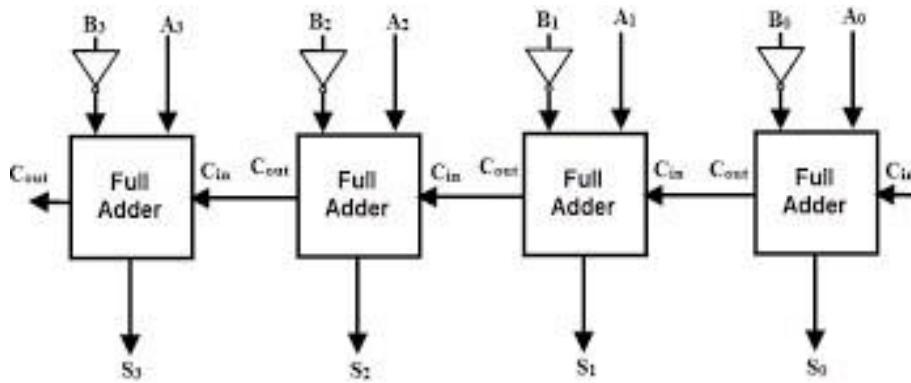
(d) $Y = 1000011$

2's complement of $X = + 0101100$

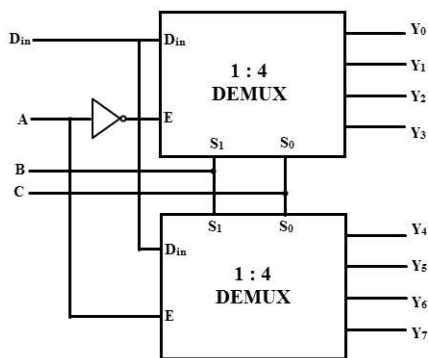
Sum= 1101111

There is no end carry. Therefore, the answer is $Y - X = -(2's \text{ complement of } 1101111) = -0010001$.

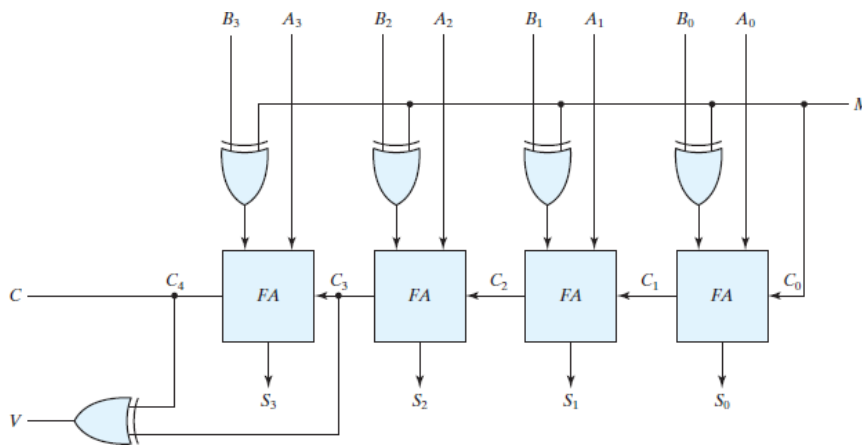
18) Draw the logic diagram of Parallel Binary Subtractor.



19) Draw 1:8 Demux using two 1:4 demux. (Nov 2018)



20) Draw the logic diagram of 2's complement adder/subtractor. (May 2013)



The mode input M controls the operation. When $M = 0$, the circuit is an adder, and when $M = 1$, the circuit becomes a subtractor.

21) What is Magnitude Comparator? [NOV – 2019]

The comparison of two numbers is an operation that determines whether one number is greater than, less than, or equal to the other number. A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes.

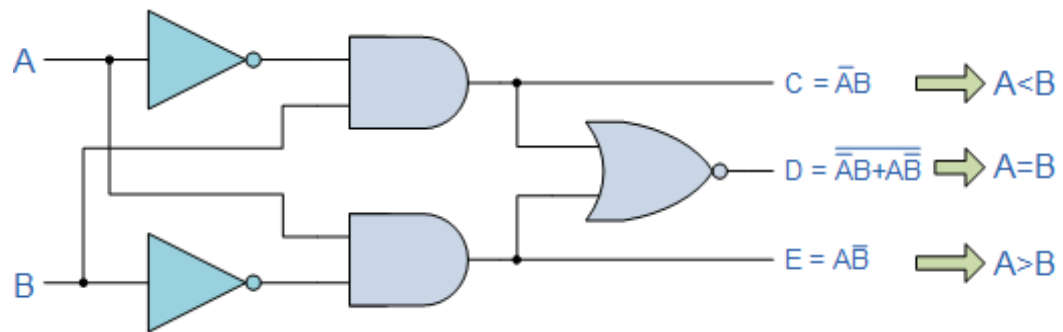
The outcome of the comparison is specified by three binary variables that indicate whether $A > B$, $A = B$, or $A < B$.

22) Design a 1-bit Magnitude Comparator.

Truth table:

Inputs		Outputs		
B	A	$A > B$	$A = B$	$A < B$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Logic Circuits:



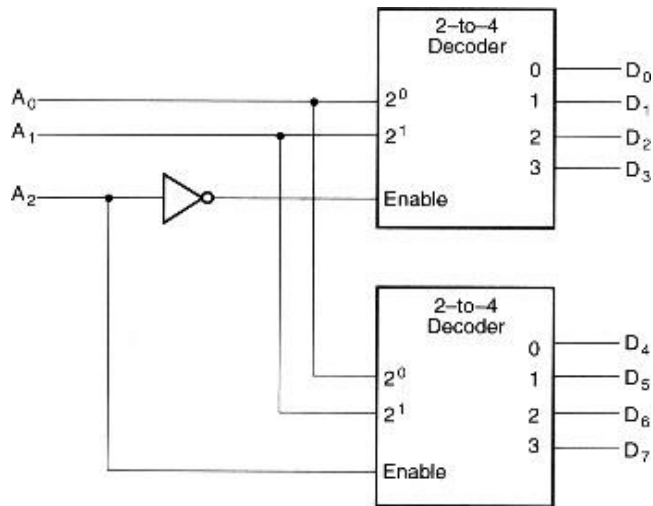
23) What is Decoder? What are binary decoders? (Nov 2017)

A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n -bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.

The purpose of a decoder is to generate the 2^n (or fewer) minterms of n input variables, shown below for two input variables.

24) Design a 3 to 8 decoder with 2 to 4 decoder.

Not that the two to four decoder design shown earlier, with its *enable* inputs can be used to build a three to eight decoder as follows.



25) What is Encoder? (May 2012)

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. The output lines, as an aggregate, generate the binary code corresponding to the input value.

26) What is Priority Encoder? (Apr 2017)

A priority encoder is an encoder circuit that includes the priority function. The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

27) Define Multiplexer (MUX) (or) Data Selector. (Dec 2006, May 2011) [NOV – 2019]

A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

28) What is De-multiplexer?

The de-multiplexer performs the inverse function of a multiplexer, that is it receives information on one line and transmits it onto one of 2^n possible output lines. The selection is by n input select lines.

29) What is Parity?

A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The message, including the parity bit, is transmitted and then checked at the receiving

end for errors. An error is detected if the checked parity does not correspond with the one transmitted.

30) What is Parity Checker / Generator:

The circuit that generates the parity bit in the transmitter is called a *parity generator*. The circuit that checks the parity in the receiver is called a *parity checker*.

31) What is even parity and odd parity?

In even parity system, the parity bit is '0' if there are even number of 1s in the data and the parity bit is '1' if there are odd number of 1s in the data.

In odd parity system, the parity bit is '1' if there are even number of 1s in the data and the parity bit is '0' if there are odd number of 1s in the data.

31) Give the applications of Demultiplexer.

- i) It finds its application in Data transmission system with error detection.
- ii) One simple application is binary to Decimal decoder.

32) Mention the uses of Demultiplexer.

Demultiplexer is used in computers when a same message has to be sent to different receivers. Not only in computers, but any time information from one source can be fed to several places.

33) Give other name for Multiplexer and Demultiplexer.

Multiplexer is otherwise called as Data selector.

Demultiplexer is otherwise called as Data distributor.

34) What is the function of the enable input in a Multiplexer?

The function of the enable input in a MUX is to control the operation of the unit.

35) List out the applications of decoder? (Dec 2006)

- a. Decoders are used in counter system.
- b. They are used in analog to digital converter.
- c. Decoder outputs can be used to drive a display system.

36) What is the Application of Mux?

- 1. They are used as a data selector to select one output of many data inputs.
- 2. They can be used to implement combinational logic circuits
- 3. They are used in time multiplexing systems.
- 4. They are used in frequency multiplexing systems.
- 5. They are used in A/D & D/A Converter.
- 6. They are used in data acquisition system.

37) List out the applications of comparators?

- a. Comparators are used as a part of the address decoding circuitry in computers to select a specific input/output device for the storage of data.
- b. They are used to actuate circuitry to drive the physical variable towards the reference value.
- c. They are used in control applications.

38) What is carry look-ahead addition?

The speed with which an addition is performed limited by the time required for the carries to propagate or ripple through all of the stage of the adder. One method of speeding up the process is by eliminating the ripple carry delay.

39) What is the Difference between Decoder & Demux.?

S.No	Decoder	Demux
1	Decoder is a many inputs to many Outputs	Demux is a single input to many outputs
2	There are no selection lines.	The selection of specific output line is controlled by the value of selection lines.

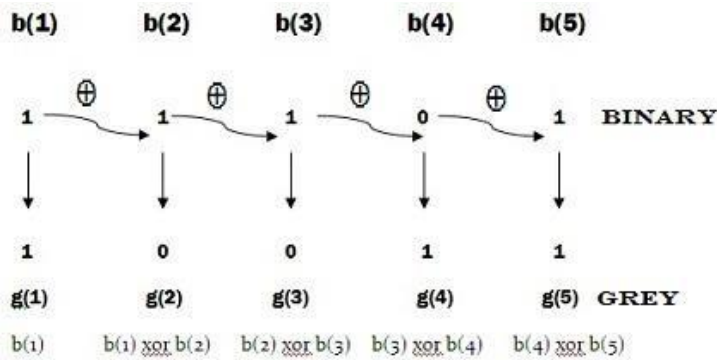
40) How Binary to Gray Code Conversion done?

Consider b_1, b_2, b_3, b_4 and b_5 is the Binary Number and it is need be converted into Grey Code.

- 1. Write Most Significant Bit (MSB) is same as the MSB in Binary Number.
- 2. The second bit of the Grey code can be found by performing the Exclusive-OR (EX-OR) operation between the First and second bits of the Binary Number.
- 3. The Third bit of the Grey code can be found by performing the Exclusive-OR (EX-OR) operation between the Third and Second bits of the given Binary Number; and so on

EX-OR Operation:

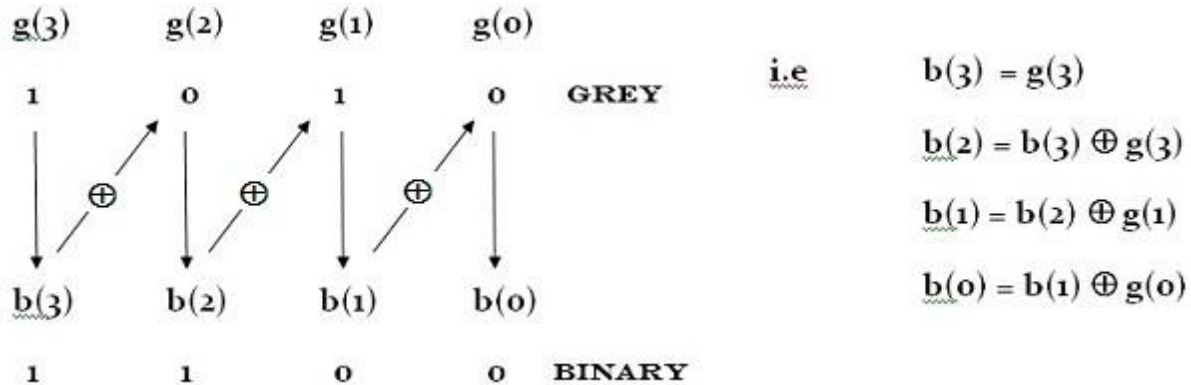
- 1. Both the bits are 0 or 1 then the output of EX-OR gate will be 0.
- 2. Any one of the bit in two bits is 1 then the output of EX-OR gate will be 1.



41) How Gray Code to Binary Conversion done?

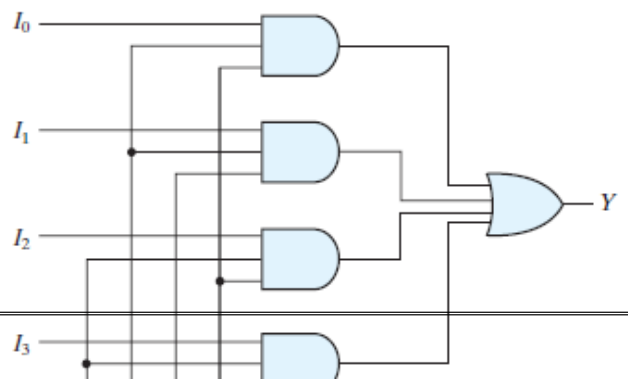
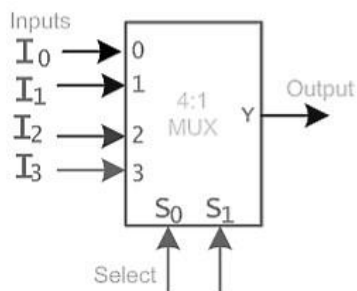
Consider g_0, g_1, g_2 and g_3 is the Gray Code and it is need be converted into Binary Number. The steps for Binary to Gray Code Conversion needs to be reversed to find out the equivalent Binary Number

1. The Most Significant Bit (MSB) of the Binary is same as the First MSB of the Gray Code.
2. If the second Gray Bit is 0 then the second bit of the Binary is bit will be same as that of the First Binary bit; if the Second Gray Bit is 1 then the Second Bit of the Binary will be inverse of its previous binary bit. Refer the below image for easy understanding of Gray to Binary Conversion



32) Draw the circuit for 4 to 1 line multiplexer. (Apr 2017) [NOV – 2019]

Logic Diagram:



Truth Table:

SELECT INPUT		OUTPUT
S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

42) What do you mean by HDL?

Hardware description language (HDL)- hardware description language or HDL is any language from a class of computer languages and/or programming languages for formal description of digital logic and electronic circuits.

43) List the Verilog HDL model of a combinational circuits.

- ✓ Gate level modeling-using instantiations of predefined and user defined primitive gates.
- ✓ Dataflow modeling using continuous assignment with the keyword **assign**.
- ✓ Behavioral modeling using procedural assignment statements with the keyword **always**.

44) What is meant by Gate level modeling?

In this type, a circuit is specified by its logic gates and their interconnections. Gate level modeling provides a textual description of a schematic diagram.

45) What is meant by data flow modeling?

Data flow modeling of combinational logic uses a number of operators that act on operands to produce desired results. Verilog HDL provides about 30 different operators.

46) What is meant by Behavioral modeling?

Behavioral modeling represents digital circuits at a functional and algorithmic level. It is used mostly to describe sequential circuits, but can also be used to describe combinational circuits.

47) What is Verilog?

Verilog is a general purpose hardware descriptor language. It has similar in syntax to the C programming language. It can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the switch level.

48) Define logic synthesis and simulation.

Logic synthesis is the process of deriving a list of components and their interconnection (called netlist) from the model of a digital system.

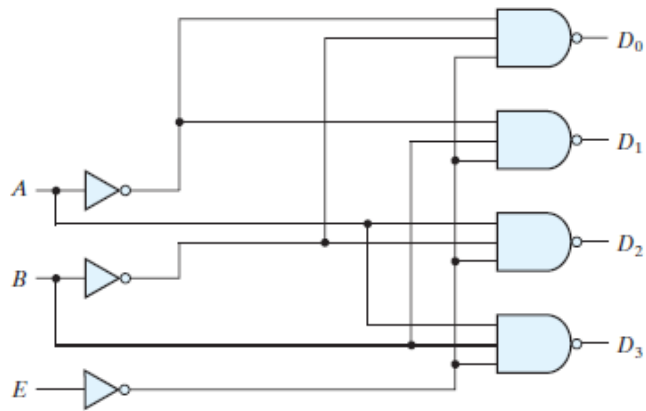
Logic Simulation is the representation of the structure and behavior of a digital logic synthesis through the use of a computer.

49) List the standard HDLs that supported by IEEE.

- ✓ VHDL (very High Speed Integrated Circuit HDL)
- ✓ Verilog HDL

50) Write the truth table of 2 to 4 line decoder and draw its logic diagram. (Apr – 2019)

2 to 4 decoder:



(a) Logic diagram

<i>E</i>	<i>A</i>	<i>B</i>	<i>D</i> ₀	<i>D</i> ₁	<i>D</i> ₂	<i>D</i> ₃
1	<i>X</i>	<i>X</i>	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1

(b) Truth table

51) State the different modeling techniques used in HDL. (Apr 2018)

- Gate level modeling
- Data flow modeling
- Behavioral modeling

UNIT III

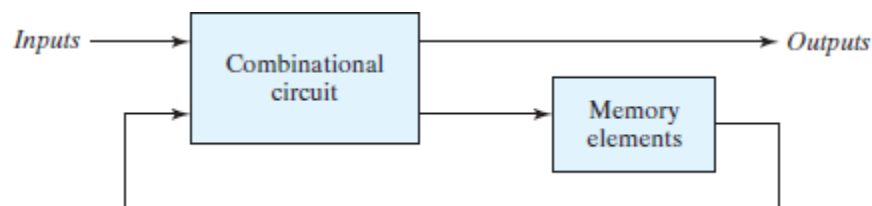
SYNCHRONOUS SEQUENTIAL LOGIC

Sequential Circuits - Storage Elements: Latches , Flip-Flops - Analysis of Clocked Sequential Circuits - State Reduction and Assignment - Design Procedure - Registers and Counters - HDL Models of Sequential Circuits

SEQUENTIAL CIRCUITS

Sequential circuits:

- Sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements.
- Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states.



Types of sequential circuits:

There are two main types of sequential circuits, and their classification is a function of the timing of their signals.

1. Synchronous sequential circuit:

It is a system whose behavior can be defined from the knowledge of its signals at discrete instants of time.

2. Asynchronous sequential circuits:

The behavior of an asynchronous sequential circuit depends upon the input signals at any instant of time and the order in which the inputs change. The storage elements commonly used in asynchronous sequential circuits are time-delay devices.

LATCHES AND FLIP FLOPS

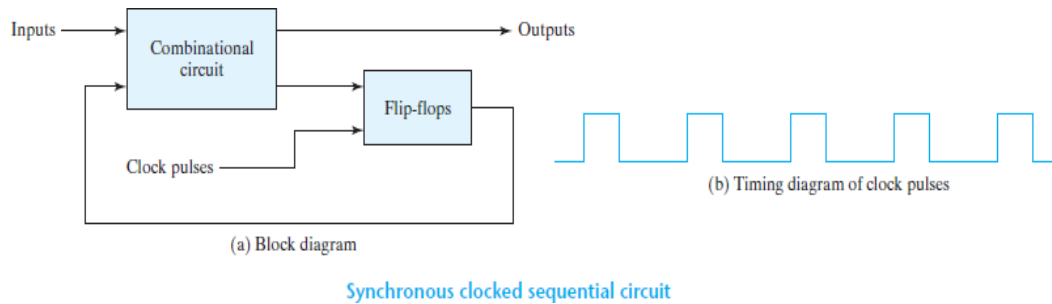
Flip-Flop:

- The storage elements (memory) used in clocked sequential circuits are called flip-flops. A flip-flop is a binary storage device capable of storing one bit of information. In a stable state, the output of a flip-flop is either 0 or 1.
- A sequential circuit may use many flip-flops to store as many bits as necessary. The block diagram of a synchronous clocked sequential circuit is shown in Fig.

- A storage element in a digital circuit can maintain a binary state indefinitely (as long as power is delivered to the circuit), until directed by an input signal to switch states.
- The major differences among various types of storage elements are in the number of inputs they possess and in the manner in which the inputs affect the binary state.

Latch:

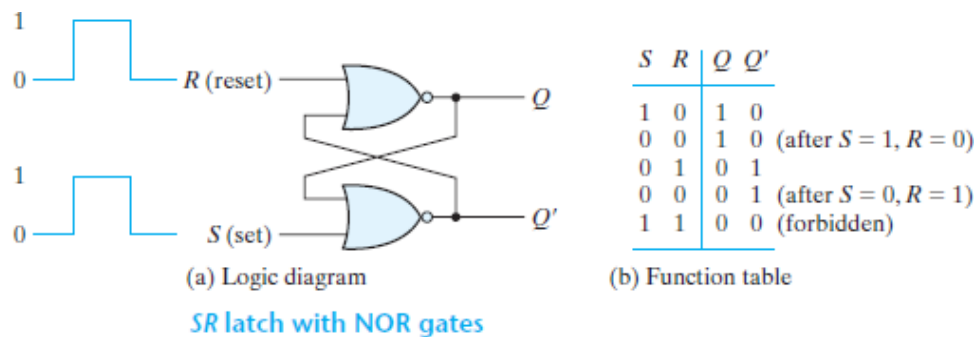
- The storage elements that operate with signal levels (rather than signal transitions) are referred to as latches; those controlled by a clock transition are flip-flops. Latches are said to be level sensitive devices; flip-flops are edge-sensitive devices.



SR Latch: Using NOR gate

Realize SR Latch using NOR and NAND gates and explain its operation.

- The SR latch is a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates, and two inputs labeled S for set and R for reset.
- The SR latch constructed with two cross-coupled NOR gates is shown in Fig.



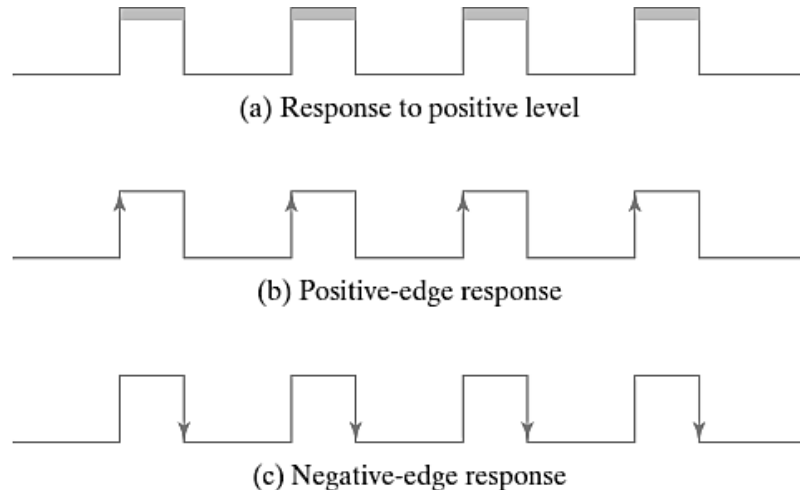
- The latch has two useful states. When output $Q = 1$ and $Q' = 0$, the latch is said to be in the *set state*. When $Q = 0$ and $Q' = 1$, it is in the *reset state*. Outputs Q and Q' are normally the complement of each other.
- However, when both inputs are equal to 1 at the same time, a condition in which both outputs are equal to 0 (rather than be mutually complementary) occurs.
- If both inputs are then switched to 0 simultaneously, the device will enter an unpredictable or undefined state or a metastable state. Consequently, in practical applications, setting both inputs to 1 is forbidden.

FLIP FLOPS

Triggering of Flip Flops:

Explain about triggering of flip flops in detail.

- The state of a latch or flip-flop is switched by a change in the control input. This momentary change is called a *trigger*, and the transition it causes is said to trigger the flip-flop.



Level Triggering:

- SR, D, JK and T latches are having enable input.
- Latches are controlled by enable signal, and they are level triggered, either positive level triggered or negative level triggered as shown in figure (a).
- The output is free to change according to the input values, when active level is maintained at the enable input.

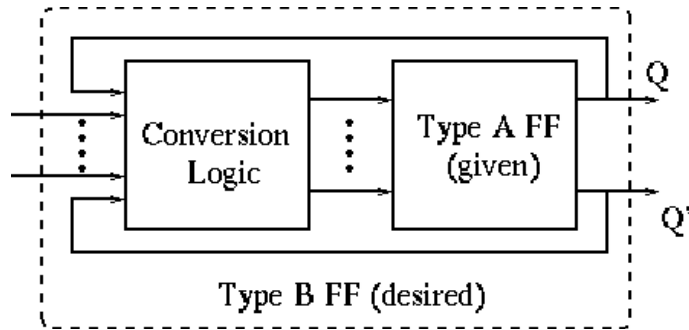
Edge Triggering:

- A clock pulse goes through two transitions: from 0 to 1 and the return from 1 to 0.
- As shown in above Fig (b) and (c), the positive transition is defined as the positive edge and the negative transition as the negative edge.

Explain the operation of flipflops.(Nov 2017)

FLIP FLOP CONVERSIONS

The purpose is to convert a given type A FF to a desired type B FF using some conversion logic.



The key here is to use the excitation table, which shows the necessary triggering signal (S,R, J,K, D and

T) for a desired flipflop state transition $Q_t \rightarrow Q_{t+1}$:

Excitation table for all flip flops:

Q_t	Q_{t+1}	S	R	D	J	K	T
0	0	0	X	0	0	X	0
0	1	1	0	1	1	X	1
1	0	0	1	0	X	1	1
1	1	X	0	1	X	0	0

1. SR Flip Flop to JK Flip Flop

The truth tables for the flip flop conversion are given below. The present state is represented by Q_p and Q_{p+1} is the next state to be obtained when the J and K inputs are applied.

For two inputs J and K, there will be eight possible combinations. For each combination of J, K and Q_p , the corresponding Q_{p+1} states are found. Q_{p+1} simply suggests the future values to be obtained by the JK flip flop after the value of Q_p .

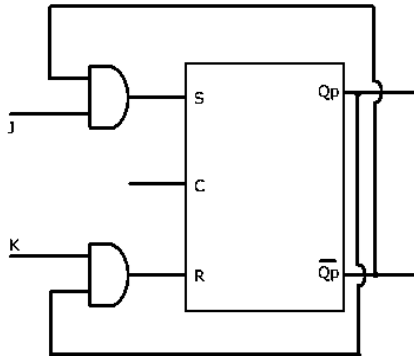
The table is then completed by writing the values of S and R required to get each Q_{p+1} from the corresponding Q_p . That is, the values of S and R that are required to change the state of the flip flop from Q_p to Q_{p+1} are written.

S-R Flip Flop to J-K Flip Flop

Conversion Table

J-K Inputs		Outputs		S-R Inputs	
J	K	Qp	Qp+1	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Logic Diagram



J	KQp	00	01	11	10
		0	1	3	2
0	0	X	0	0	0
1	1	X	0	1	1

$$S = \bar{J}Q_p$$

J	KQp	00	01	11	10
		0	1	3	2
0	X	0	1	X	X
1	0	0	1	0	0

$$R = KQ_n$$

2. JK Flip Flop to SR Flip Flop

This will be the reverse process of the above explained conversion. S and R will be the external inputs to J and K. As shown in the logic diagram below, J and K will be the outputs of the combinational circuit. Thus, the values of J and K have to be obtained in terms of S, R and Qp. The logic diagram is shown below.

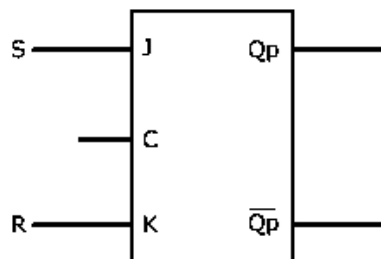
A conversion table is to be written using S, R, Qp, Qp+1, J and K. For two inputs, S and R, eight combinations are made. For each combination, the corresponding Qp+1 outputs are found. The outputs for the combinations of S=1 and R=1 are not permitted for an SR flip flop. Thus the outputs are considered invalid and the J and K values are taken as “don’t cares”.

J-K Flip Flop to S-R Flip Flop

Conversion Table

S-R Inputs		Outputs		J-K Inputs	
S	R	Qp	Qp+1	J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	Invalid		Dont care	
1	1	Invalid		Dont care	

Logic Diagram



S	RQp	00	01	11	10
		0	1	3	2
0	0	X	X	0	0
1	1	X	X	X	X

$$J=S$$

S	RQp	00	01	11	10
		0	1	3	2
0	X	0	1	X	X
1	X	0	X	X	X

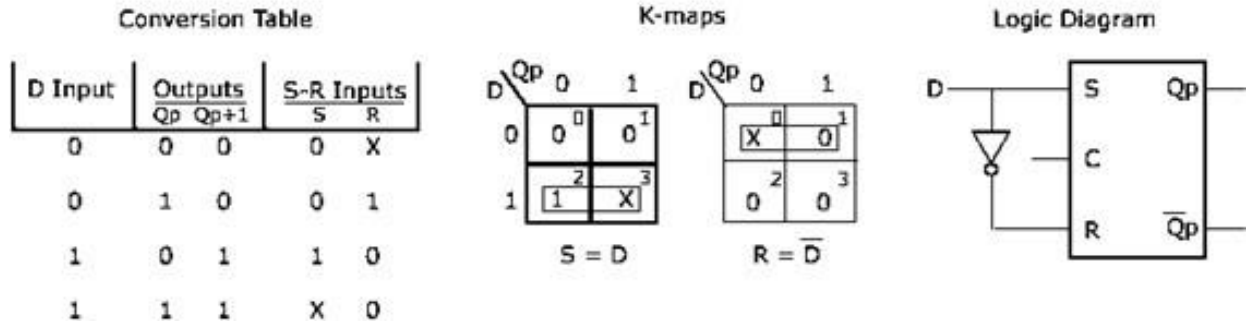
$$K=R$$

$$K=R$$

3. SR Flip Flop to D Flip Flop

As shown in the figure, S and R are the actual inputs of the flip flop and D is the external input of the flip flop. The four combinations, the logic diagram, conversion table, and the K-map for S and R in terms of D and Qp are shown below.

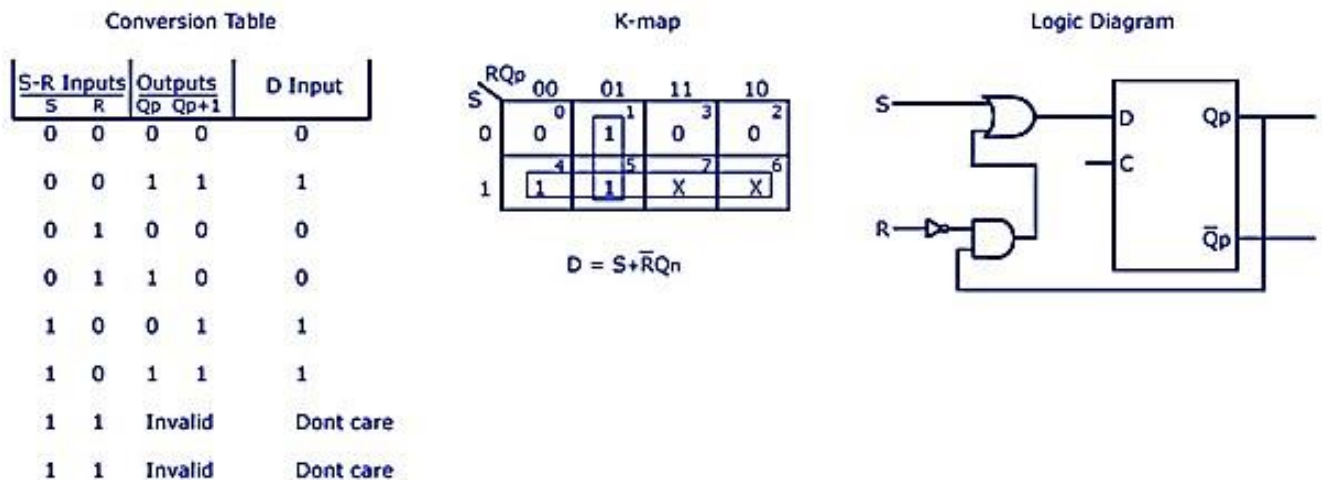
S-R Flip Flop to D Flip Flop



4. D Flip Flop to SR Flip Flop

D is the actual input of the flip flop and S and R are the external inputs. Eight possible combinations are achieved from the external inputs S, R and Qp. But, since the combination of S=1 and R=1 are invalid, the values of Qp+1 and D are considered as “don’t cares”. The logic diagram showing the conversion from D to SR, and the K-map for D in terms of S, R and Qp are shown below.

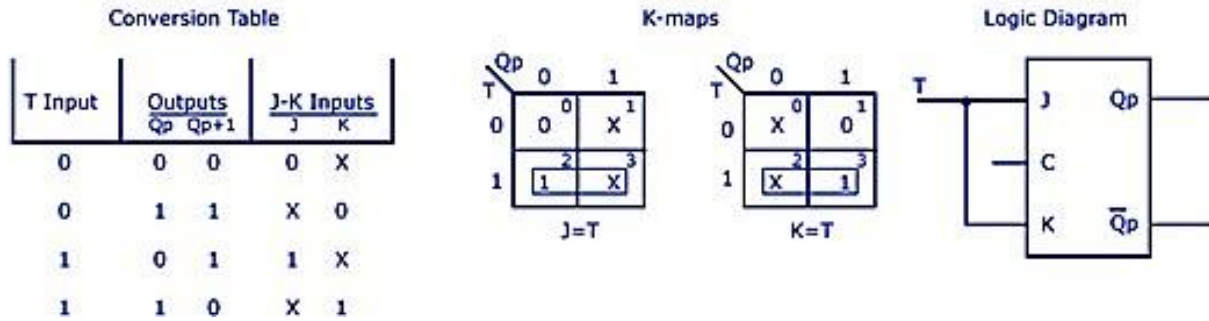
D Flip Flop to S-R Flip Flop



5. JK Flip Flop to T Flip Flop

J and K are the actual inputs of the flip flop and T is taken as the external input for conversion. Four combinations are produced with T and Qp. J and K are expressed in terms of T and Qp. The conversion table, K-maps, and the logic diagram are given below.

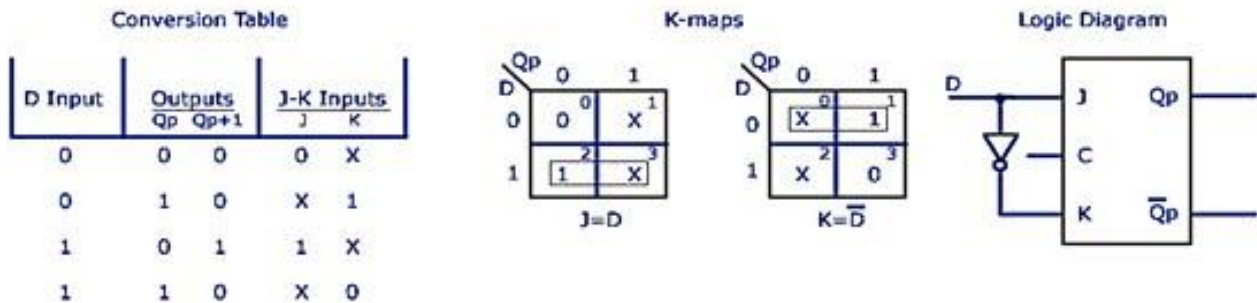
J-K Flip Flop to T Flip Flop



6. JK Flip Flop to D Flip Flop

D is the external input and J and K are the actual inputs of the flip flop. D and Qp make four combinations. J and K are expressed in terms of D and Qp. The four combination conversion table, the K-maps for J and K in terms of D and Qp, and the logic diagram showing the conversion from JK to D are given below.

J-K Flip Flop to D Flip Flop



7. D Flip Flop to JK Flip Flop

AUQ: How will you convert a D flip-flop into JK flip-flop? (AUQ: Dec 2009,11, Apr 2017)

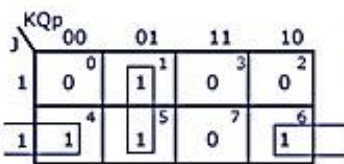
In this conversion, D is the actual input to the flip flop and J and K are the external inputs. J, K and Qp make eight possible combinations, as shown in the conversion table below. D is expressed in terms of J, K and Qp. The conversion table, the K-map for D in terms of J, K and Qp and the logic diagram showing the conversion from D to JK are given in the figure below.

D Flip Flop to J-K Flip Flop

Conversion Table

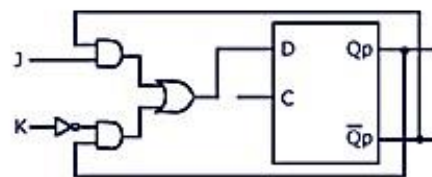
J-K Input		Outputs		D Input
J	K	Q_p	Q_{p+1}	
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

K-map



$$D = J\bar{Q}_p + \bar{K}Q_p$$

Logic Diagram



MEALY AND MOORE MODELS

Write short notes on Mealy and Moore models in sequential circuits.

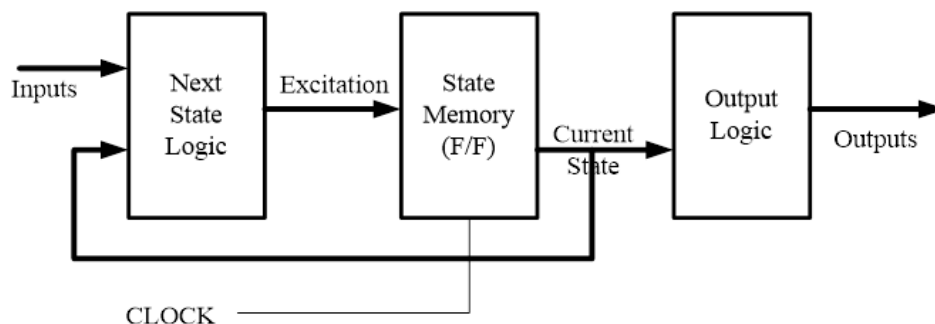
- In synchronous sequential circuit the outputs depend upon the order in which its input variables change and can be affected at discrete instances of time.

General Models:

- There are two models in sequential circuits. They are:
 1. Mealy model
 2. Moore model

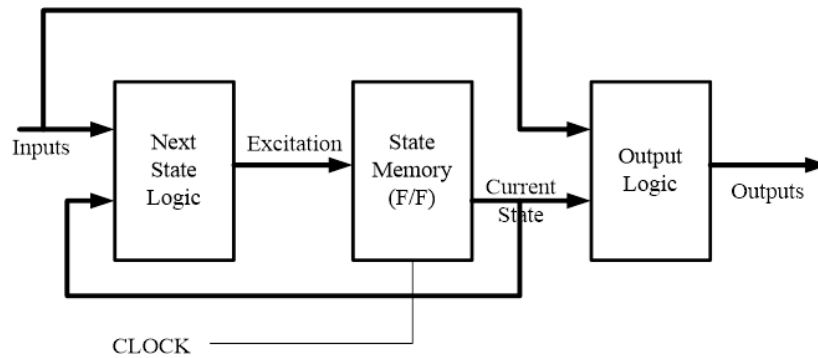
Moore machine:

- In the Moore model, the outputs are a function of present state only.



Mealy machine:

- In the Mealy model, the outputs are a function of present state and external inputs.



Difference between Moore model and Mealy model.

Sl.No	Moore model	Mealy model
1	Its output is a function of present state only.	Its output is a function of present state as well as present input.
2	Input changes does not affect the output.	Input changes may affect the output of the circuit.
3	It requires more number of states for implementing same function.	It requires less number of states for implementing same function.

Example:

A sequential circuit with two 'D' Flip-Flops A and B, one input (x) and one output (y).

The Flip-Flop input functions are:

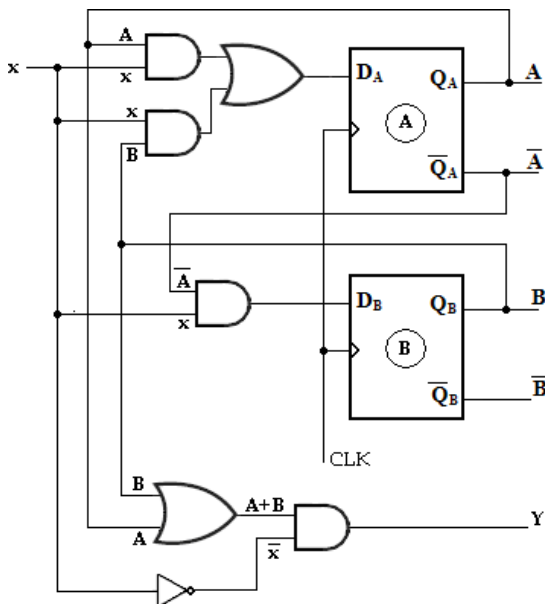
$D_A = Ax + Bx$

$D_B = A'x$ and

the circuit output function is, $Y = (A + B) x'$.

(a) Draw the logic diagram of the circuit, (b) Tabulate the state table, (c) Draw the state diagram.

Solution:



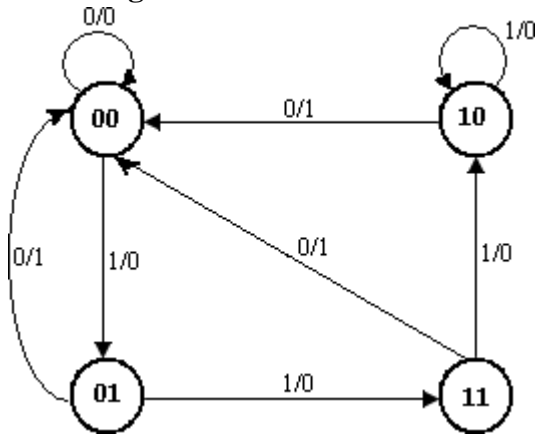
State table:

Present state		Input	Flip-Flop Inputs		Next state		Output
A	B	x	$D_A = Ax+Bx$	$D_B = A'x$	A(t+1)	B(t+1)	$Y = (A+B)x'$
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	0	0	0	0	1
0	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	1
1	1	1	1	0	1	0	0

Present state		Next state				Output	
		x=0		x=1		x=0	x=1
A	B	A	B	A	B	Y	Y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

Second form of state table

State diagram:



TWO MARKS

1. Difference between Combinational & Sequential Circuits.

S.no	Combinational Circuits	Sequential Circuits
1	The output at all times depends only on the present combination of input variables.	The output not only depends on the present input but also depends on the past history input variables.
2	Memory unit is not Required	Memory unit is required to store the past history of input variable
3	Clock input is not needed.	Clock input is needed.

4	Faster in Speed	Speed is Slower
5	Easy to design. Eg: Mux, Demux, Encoder, Decoder, Adders, Subtractors.	Difficult to design. Eg: Shift Register, Counters.

2. What are the classifications of sequential circuits?

The sequential circuits are classified on the basis of timing of their signals into two types. They are
1) Synchronous sequential circuit. 2) Asynchronous sequential circuit.

3. Define Latch.

The basic unit for storage is Latch. A Latch maintains its output state either at 1 or 0 until directed by an input signal to change its state.

4. Define a flip flop.

A flip-flop is a storage device capable of storing one bit of information. It has two states either 0 or 1. It is also called bistable multivibrator.

5. What are the different types of flip-flop?

The various types of flip flops are 1). SR flip-flop 2). D flip-flop 3). JK flip-flop 4). T flip-flop

6. What is the main difference between a latch and flip flop?

- ✓ The output of latch changes immediately when its input changes.
- ✓ The output of a flip-flop changes only when its clock pulse is active and its input changes. Input changes do not affect output if its clock is not activated.

7. State few applications of Flip-Flop.

- ✓ Used as a memory element.
- ✓ Used as delay elements.
- ✓ Data transfer
- ✓ Used as a building block in sequential circuits such as counters and registers.

8. What is the operation of D flip-flop?

In D flip-flop during the occurrence of clock pulse if $D=1$, the output Q is set and if $D=0$, the output is reset. Set – 1, Reset – 0.

9. What is the operation of JK flip-flop?

When K input is low and J input is high the Q output of flip-flop is set.

When K input is high and J input is low the Q output of flip-flop is reset.

When both the inputs K and J are low the output does not change

When both the inputs K and J are high it is possible to set or reset the flip-flop (ie) the output toggles on the next positive clock edge.

10. What is the operation of T flip-flop? (Nov 2018)

T flip-flop is also known as Toggle flip-flop. 1). When $T=0$ there is no change in the output. 2). When $T=1$ the output switch to the complement state (ie) the output toggles.

11. Define race around condition.

In JK flip-flop output is fed back to the input. Therefore change in the output results change in the input. Due to this in the positive half of the clock pulse if both J and K are high then output toggles continuously. This condition is called 'race around condition'.

12. What is triggering? What is the need for trigger in flip-flop?

A flip-flop is made to change its state by application of a clock pulse after giving inputs. This is called triggering. The clock (triggering input) is given to synchronize the change in the output with it.

13. What is meant by level and edge-triggering? (Nov 2017) (Apr – 2019)

- ✓ If flip-flop changes its state when the clock is positive (high) or negative (low) then, that flip-flop is said to be *level triggering flip-flop*.
- ✓ If the flip-flop changes its state at the positive edge (rising edge) or negative edge (falling edge) of the clock is sensitive to its inputs only at this transition of the clock then flip-flop is said to be *edge triggered flip-flop*.

14. How do you eliminate race around condition in JK flip flop. ?

Using master-slave flip-flop which consists of two flip-flops where one circuit serves as a master and the other as a slave race around condition in JK flip flop is eliminated .

15. Define rise time.

The time required to change the voltage level from 10% to 90% is known as rise time (t_r).

16. Define fall time.

The time required to change the voltage level from 90% to 10% is known as falltime (t_f).

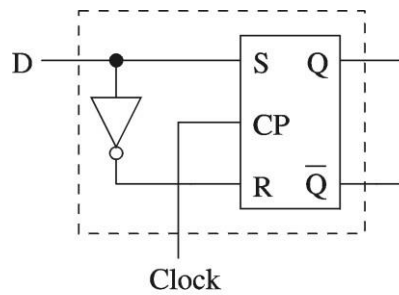
17. Define skew and clock skew.

The phase shift between the rectangular clock waveforms is referred to as skew and the time delay between the two clock pulses is called clock skew.

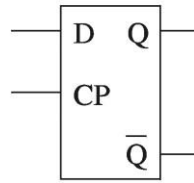
18. Define setup time.

The setup time is the minimum time required to maintain a constant voltage levels at the excitation inputs of the flip-flop device prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip flop.

19. Draw the logic diagram and write the function table of D Latch. (Apr 2019)



(a) Circuit diagram



(b) Logic symbol

D	Q_{n+1}
0	0
1	1

(c) Truth table

20. Define hold time.

The hold time is the minimum time for which the voltage levels at the excitation inputs must remain constant after the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip flop.

21. Define propagation delay.

A propagation delay is the time required to change the output after the application of the input

22. Explain the flip-flop excitation tables for RS FF.

In RS flip-flop there are four possible transitions from the present state to the Next state. They are

- 1). $0 \rightarrow 0$ transition: This can happen either when $R=S=0$ or when $R=1$ and $S=0$.
- 2). $0 \rightarrow 1$ transition: This can happen only when $S=1$ and $R=0$.
- 3). $1 \rightarrow 0$ transition: This can happen only when $S=0$ and $R=1$.
- 4). $1 \rightarrow 1$ transition: This can happen either when $S=1$ and $R=0$ or $S=0$ and $R=0$.

23. Give some applications of clocked RS Flip-flop.

Clocked RS flip flops are used in Calculators & Computers.

It is widely used in modern electronic products.

24. What is the drawback of SR Flipflop? How is this minimized? (Apr 2018)

In SR flipflop when both S and R inputs are one it will generate a Undetermined state. This is Minimized by providing feedback path or by using JK flip flop.

25. How many flip flops are required to build a Binary counter that counts from 0 to 1023?

$2^{10} = 1024$ hence 10 flipflops are required.

26. State the difference between latches and flipflops. (Apr 2019)

Latches	Flip Flops
Latches are building blocks of sequential circuits and these can be built from logic gates	Flip flops are also building blocks of sequential circuits. But, these can be built from the latches.
Latch continuously checks its inputs and changes its output correspondingly.	Flip flop continuously checks its inputs and changes its output correspondingly only at times determined by clocking signal
The latch is sensitive to the duration of the pulse and can send or receive the data when the switch is on	Flipflop is sensitive to a signal change. They can transfer data only at the single instant and data cannot be changed until next signal change. Flip flops are used as a register.
It is based on the enable function input	It works on the basis of clock pulses
It is a level triggered, it means that the output of the present state and input of the next state depends on the level that is binary input 1 or 0.	It is an edge triggered, it means that the output and the next state input changes when there is a change in clock pulse whether it may a +ve or -ve clock pulse.

27. What is mealy and Moore circuit? Or what are the models used to represent clocked sequential circuits?

- ✓ *Mealy circuit* is a network where the output is a function of both present state and input.
- ✓ *Moore circuit* is a network where the output is function of only present state

COUNTERS

Counter:

- A counter is a register (group of Flip-Flop) capable of counting the number of clock pulse arriving at its clock input.
- A counter that follows the binary number sequence is called a binary counter.
- Counter are classified into two types,
 1. Asynchronous (Ripple) counters.
 2. Synchronous counters.
- In ripple counter, a flip- flop output transition serves as clock to next flip-flop.

- With an asynchronous circuit, all the bits in the count do not all change at the same time.
- In a synchronous counter, all flip-flops receive common clock.
 - With a synchronous circuit, all the bits in the count change synchronously with the assertion of the clock
- A counter may count up or count down or count up and down depending on the input control.

Uses of Counters:

The most typical uses of counters are

- ✓ To count the number of times that a certain event takes place; the occurrence of event to be counted is represented by the input signal to the counter
- ✓ To control a fixed sequence of actions in a digital system
- ✓ To generate timing signals
- ✓ To generate clocks of different frequencies

Modulo 16 ripple /Asynchronous Up Counter

Explain the operation of a 4-bit binary ripple counter.

- The output of up-counter is incremented by one for each clock transition.
- A 4-bit asynchronous up-counter consists of 4JK Flip-Flops.
- The external clock signal is connected to the clock input of the first FlipFlop.
- The clock inputs of the remaining Flip-Flops are triggered by the Q output of the previous stage.
- We know that in JK Flip-Flop, if $J=1$, $K=1$ and clock is triggered the past output will be complemented.
- Initially, the register is cleared, $Q_DQ_CQ_BQ_A = 0000$.
- During the first clock pulse, Flip-Flop A triggers, therefore $Q_A=1$, $Q_B=Q_C=Q_D=0$.

$$Q_DQ_CQ_BQ_A=0001$$

- At the second clock pulse FLipFlop A triggers, therefore Q_A changes from 1 to 0, which triggers FlipFlop B, therefore $Q_B=1, Q_A=Q_C=Q_D=0$

$$Q_DQ_CQ_BQ_A=0010$$

- At the third clock pulse FlipFlop A triggers, therefore Q_A changes from 0 to 1, This never triggers FlipFlop B because 0 to 1 transition gives a positive edge triggering, but here the FlipFlops are triggered only at negative edge(1 to 0 transition) therefore $Q_A=Q_B=1$, $Q_C=Q_D=0$.

$$Q_DQ_CQ_BQ_A=0011$$

- At the fourth clock pulse Flip-Flop A triggers, therefore Q_A changes from 1 to 0, This triggers FlipFlop B therefore Q_B changes from 1 to 0. The change in Q_B from 1 to 0 triggers C Flip-Flop,

➤ Therefore Q_C changes from 0 to 1. Therefore $Q_A=Q_B=Q_D=0$, $Q_C=1$.

$$Q_D Q_C Q_B Q_A = 0100$$

Truth table:

CLK	Outputs			
	Q _D	Q _C	Q _B	Q _A
-	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Truth table for 4-bit asynchronous up-counter

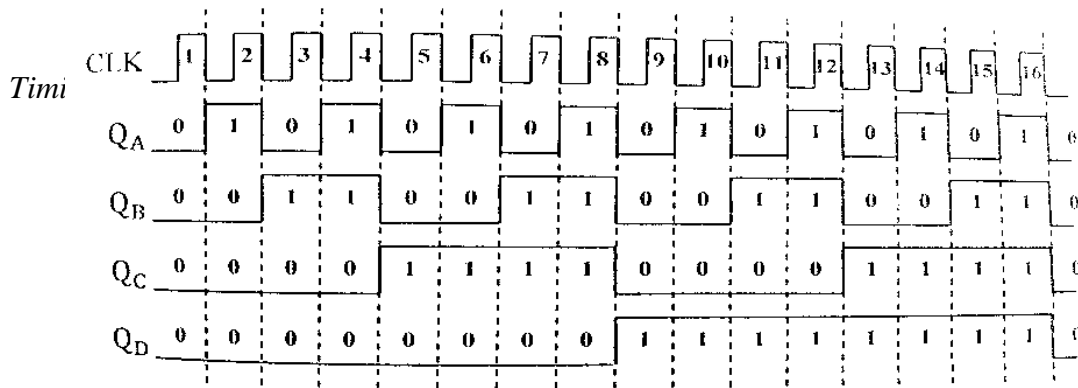


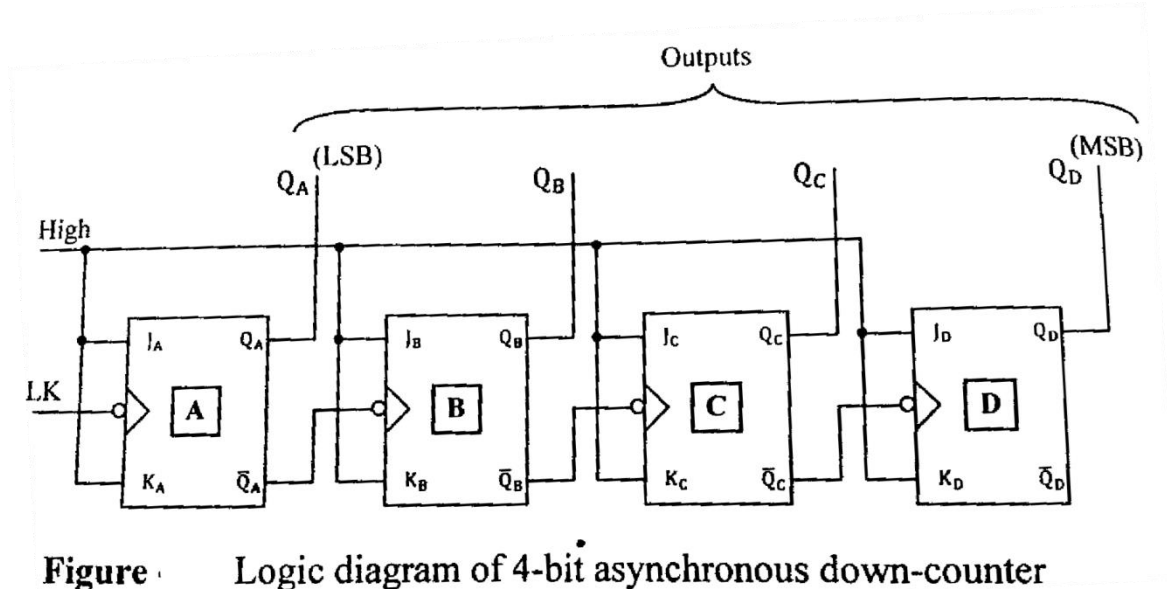
Figure 4.37 Timing diagram of 4-bit asynchronous up-counter.

Modulo 16 / 4 bit Ripple Down counter/ Asynchronous Down counter

Explain about Modulo 16 / 4 bit Ripple Down counter.

- The output of down-counter is decremented by one for each clock transition.
- A 4-bit asynchronous down-counter consists of 4JK Flip-Flops.

- The external clock signal is connected to the clock input of the first Flip-Flop.
- The clock inputs of the remaining Flip-Flops are triggered by the \overline{Q} output of the previous stage.
- We know that in JK Flip-Flop, if $J=1$, $K=1$ and clock is triggered the past output will be complemented.



- Initially, the register is cleared, $Q_D Q_C Q_B Q_A = 0000$.
- During the first clock pulse, Flip-Flop A triggers, therefore Q_A changes from 0 to 1 also $\overline{Q_A}$ changes from 1 to 0. This triggers Flip-Flop B, therefore Q_B changes from 0 to 1, also $\overline{Q_B}$ changes from 1 to 0 which triggers Flip-Flop C. Hence Q_C changes from 0 to 1 and $\overline{Q_C}$ changes from 1 to 0, which further triggers, Flip-Flop D.

$$Q_D Q_C Q_B Q_A = 1111$$

$$\overline{Q_D} \overline{Q_C} \overline{Q_B} \overline{Q_A} = 0000$$

- During the second clock pulse Flip-Flop A triggers, therefore Q_A changes from 1 to 0 also $\overline{Q_A}$ changes from 0 to 1 which never triggers B Flip-Flop. Therefore C and D Flip-Flop are not triggered.

$$Q_D Q_C Q_B Q_A = 1110$$

- The same procedure repeats until the counter decrements upto 0000.

CLK	Outputs			
	Q _D	Q _C	Q _B	Q _A
-	0	0	0	0
1	1	1	1	1
2	1	1	1	0
3	1	1	0	1
4	1	1	0	0
5	1	0	1	1
6	1	0	1	0
7	1	0	0	1
8	1	0	0	0
9	0	1	1	1
10	0	1	1	0
11	0	1	0	1
12	0	1	0	0
13	0	0	1	1
14	0	0	1	0
15	0	0	0	1
16	0	0	0	0

Table ... Truth table for 4-bit asynchronous down-counter

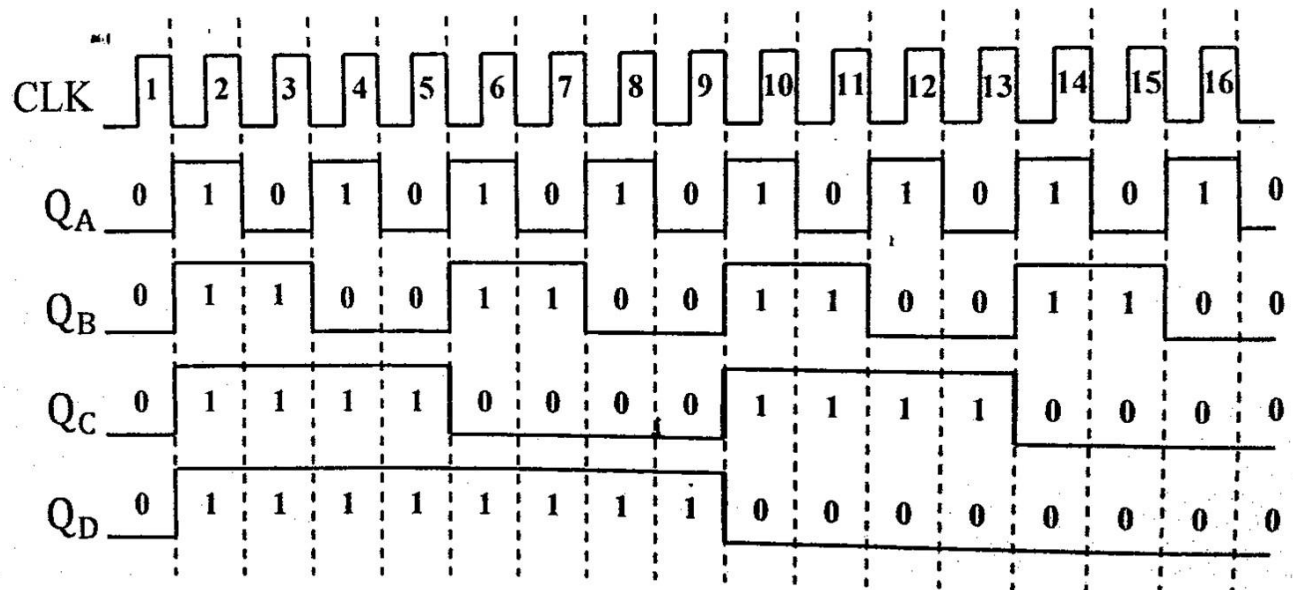


Figure 4 Timing diagram of 4-bit asynchronous down-counter.

Asynchronous Up/Down Counter:

Explain about Asynchronous Up/Down counter.

- The up-down counter has the capability of counting upwards as well as downwards. It is also called multimode counter.
- In asynchronous up-counter, each flip-flop is triggered by the normal output Q of the preceding flip-flop.
- In asynchronous down counter, each flip-flop is triggered by the complement output \bar{Q} of the preceding flip-flop.
- In both the counters, the first flip-flop is triggered by the clock output.

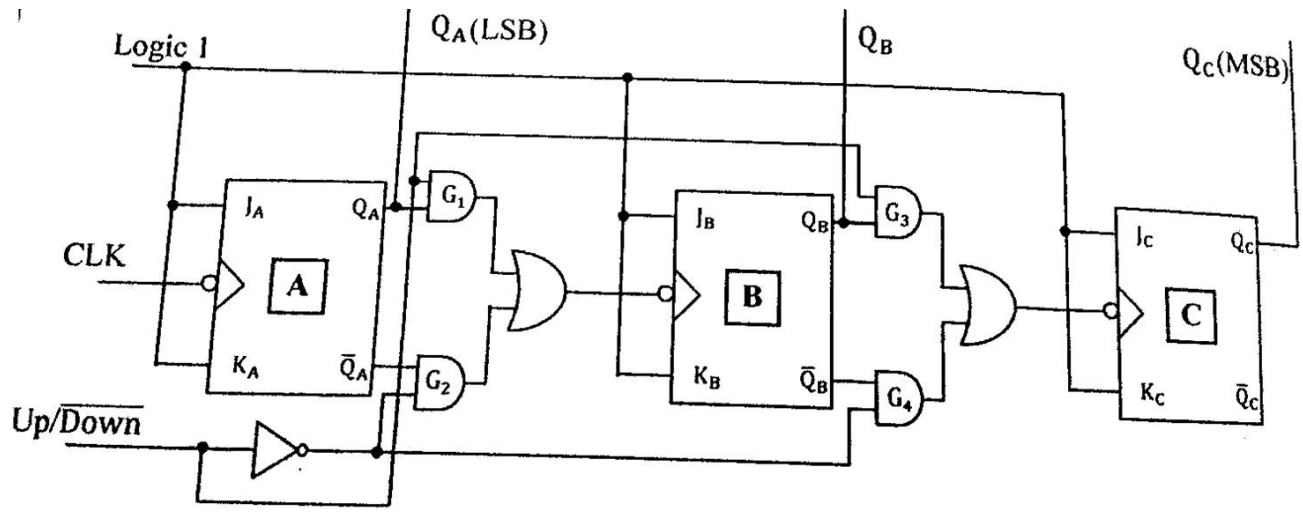


Figure 3-bit asynchronous up/down-counter

- If $\overline{Up/Down} = 1$, the 3-bit asynchronous up/down counter will perform up-counting. It will count from 000 to 111. If $\overline{Up/Down} = 1$ gates G_2 and G_4 are disabled and gates G_1 and G_3 are enabled. So that the circuit behaves as an up-counter circuit.
- If $\overline{Up/Down} = 0$, the 3-bit asynchronous up/down counter will perform down-counting. It will count from 111 to 000. If $\overline{Up/Down} = 0$ gates G_2 and G_4 are enabled and gates G_1 and G_3 are disabled. So that the circuit behaves as a down-counter circuit.

$\overline{Up/Down}$	Q_C	Q_B	Q_A
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1
0	1	1	1
0	1	0	1
0	1	0	0
0	0	1	0
0	0	1	1
0	0	0	1
0	0	0	0

Table : Truth table for 3-Bit asynchronous Up/Down-counter

4-bit Synchronous up-counter:

Explain about 4-bit Synchronous up-counter.

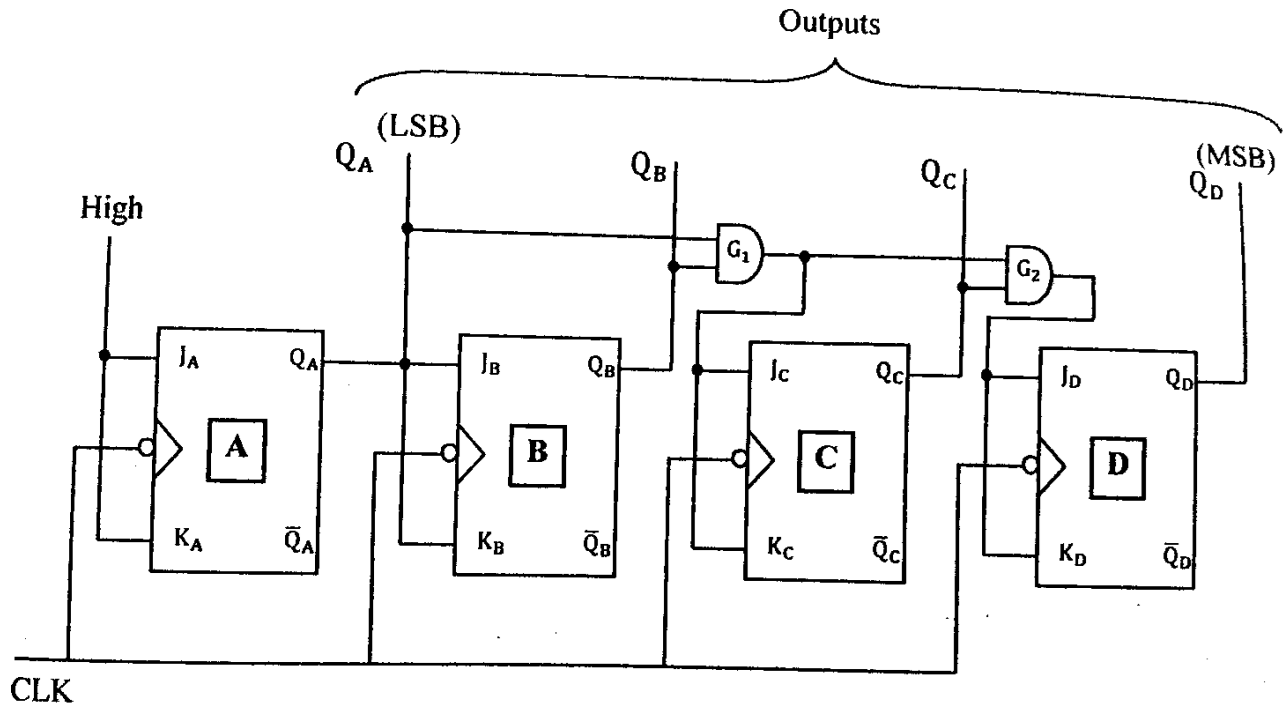


Figure Logic diagram of 4-bit Synchronous up-counter

- In JK Flip-Flop, If $J=0$, $K=0$ and clock is triggered, the output never changes. If $J=1$ and $K=1$ and the clock is triggered, the past output will be complemented.

Initially the register is cleared $Q_D Q_C Q_B Q_A = 0000$.

During the first clock pulse, $J_A = K_A = 1$, Q_A becomes 1, Q_B, Q_C, Q_D remains 0.

$$Q_D Q_C Q_B Q_A = 0001.$$

During second clock pulse, $J_A = K_A = 1$, $Q_A = 0$.

$$J_B = K_B = 1, Q_B = 1, Q_C, Q_D \text{ remains } 0.$$

$$Q_D Q_C Q_B Q_A = 0010.$$

During third clock pulse, $J_A = K_A = 1$, $Q_A = 1$.

$$J_B = K_B = 0, Q_B = 1, Q_C, Q_D \text{ remains } 0.$$

$$Q_D Q_C Q_B Q_A = 0011.$$

During fourth clock pulse, $J_A = K_A = 1$, $Q_A = 0$.

$$J_B = K_B = 1, Q_B = 0$$

$$J_C = K_C = 1, Q_C = 1$$

$$Q_D \text{ remains } 0$$

$$Q_D Q_C Q_B Q_A = 0100.$$

The same procedure repeats until the counter counts up to 1111.

CLK	Outputs			
	Q _D	Q _C	Q _B	Q _A
-	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

Table Truth table for 4-bit synchronous up-counter

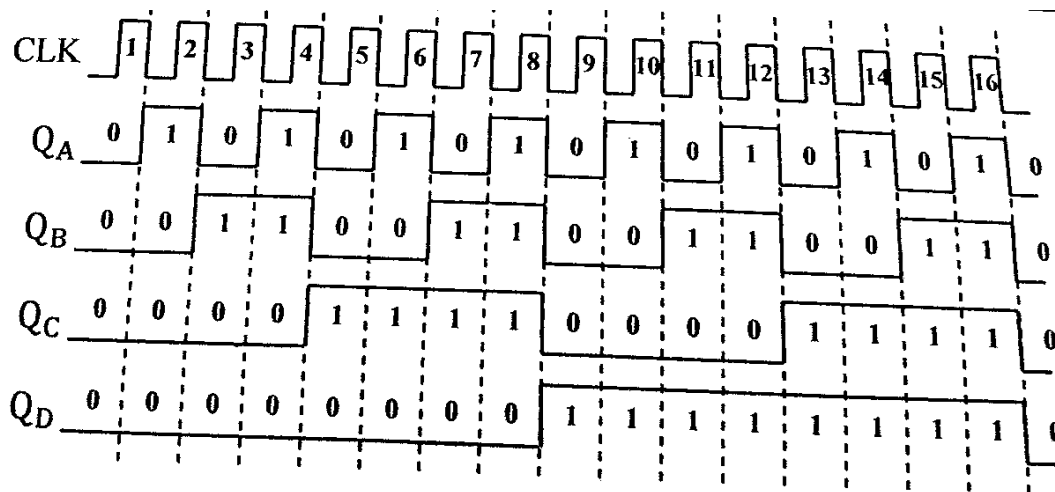


Figure Timing diagram of 4-bit synchronous up-counter

4-bit Synchronous down-counter:

Explain about 4-Bit Synchronous down counter.

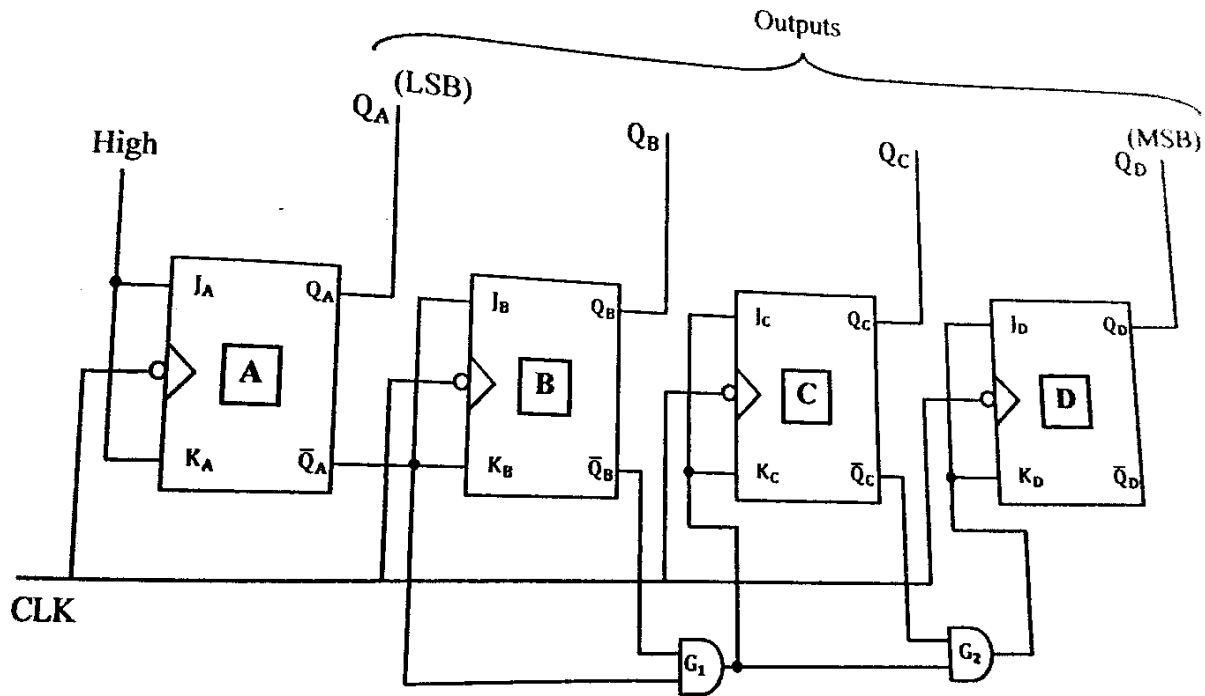


Figure 3 Logic diagram of 4-bit synchronous down-counter

In JK Flip-Flop, If $J=0$, $K=0$ and clock is triggered, the output never changes. If $J=1$ and $K=1$ and the clock is triggered, the past output will be complemented.

Initially the register is cleared $Q_D Q_C Q_B Q_A = 0000$

$$\overline{Q_D} \overline{Q_C} \overline{Q_B} \overline{Q_A} = 1111$$

During the first clock pulse, $J_A = K_A = 1$, $Q_A = 1$

$$J_B = K_B = 1, Q_B = 1$$

$$J_C = K_C = 1, Q_C = 1$$

$$J_D = K_D = 1, Q_D = 1$$

$$Q_D Q_C Q_B Q_A = 1111$$

$$\overline{Q_D} \overline{Q_C} \overline{Q_B} \overline{Q_A} = 0000$$

During the second clock pulse, $J_A = K_A = 1$, $Q_A = 0$

$$J_B = K_B = 0, Q_B = 1$$

$$J_C = K_C = 0, Q_C = 1$$

$$J_D = K_D = 0, Q_D = 1$$

$$Q_D Q_C Q_B Q_A = 1110$$

$$\overline{Q_D} \overline{Q_C} \overline{Q_B} \overline{Q_A} = 0001$$

During the second clock pulse, $J_A = K_A = 1$, $Q_A = 1$

$J_B = K_B = 1$, $Q_B = 0$

$J_C = K_C = 0$, $Q_C = 1$

$J_D = K_D = 0$, $Q_D = 1$

$Q_D Q_C Q_B Q_A = 1101$

The process repeats until the counter down-counts up to 0000.

CLK	Outputs			
	Q_D	Q_C	Q_B	Q_A
-	0	0	0	0
1	1	1	1	1
2	1	1	1	0
3	1	1	0	1
4	1	1	0	0
5	1	0	1	1
6	1	0	1	0
7	1	0	0	1
8	1	0	0	0
9	0	1	1	1
10	0	1	1	0
11	0	1	0	1
12	0	1	0	0
13	0	0	1	1
14	0	0	1	0
15	0	0	0	1
16	0	0	0	0

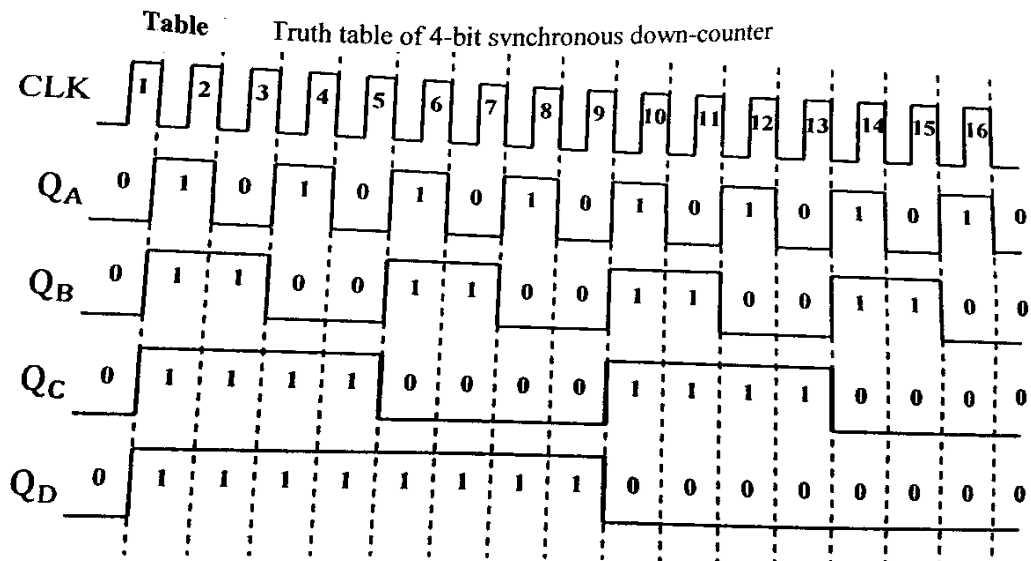


Figure ... Timing diagram of 4-bit synchronous down-counter

Modulo 8 Synchronous Up/Down Counter:

Explain about Modulo 8 Synchronous Up/Down Counter.

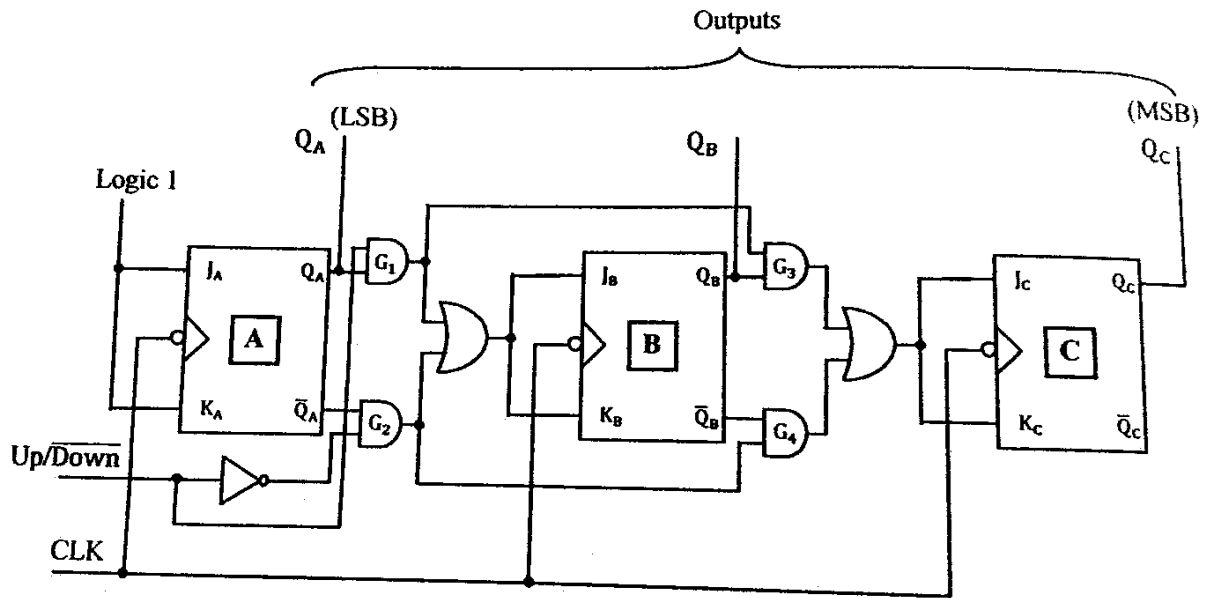


Figure ; 3-bit synchronous up/down-counter

In synchronous up-counter the $\overline{Q_A}$ output is given to J_B, K_B and $\overline{Q_A}, Q_B$ is given to J_C, K_C . But in synchronous down-counter Q_A output is given to J_B, K_B and Q_A, Q_B is given to J_C, K_C .

A control input $\overline{Up/Down}$ is used to select the mode of operation.

If $\overline{Up/Down} = 1$, the 3-bit asynchronous up/down counter will perform up-counting. It will count from 000 to 111. If $\overline{Up/Down} = 1$ gates G_2 and G_4 are disabled and gates G_1 and G_3 are enabled. So that the circuit behaves as an up-counter circuit.

If $\overline{Up/Down} = 0$, the 3-bit asynchronous up/down counter will perform down-counting. It will count from 111 to 000. If $\overline{Up/Down} = 0$ gates G_2 and G_4 are enabled and gates G_1 and G_3 are disabled. So that the circuit behaves as a down-counter circuit.

$\overline{Up/Down} = 1$	Q_C	Q_B	Q_A	$\overline{Up/Down} = 0$
	0	0	0	
	0	0	1	
	0	1	0	
	0	1	1	
	1	0	0	
	1	0	1	
	1	1	0	
	1	1	1	

Table ; Truth table for 3-Bit asynchronous Up/Down-counter

1. What is counter?

A counter is a register (group of Flip-Flop) capable of counting the number of clock pulse arriving at its clock input.

2. What is binary counter?

A counter that follows the binary number sequence is called a binary counter.

3. State the applications of counters.

1. Used as a memory Element.
2. Used as a Delay Element.
3. Used as a basic building block in sequential circuits such as counters and registers.
4. Used for Data Transfer, Frequency Division & Counting.

4. List the types of counters.

Counter are classified into two types,

- ✓ Asynchronous (Ripple) counters.
- ✓ Synchronous counters.

5. Give the comparison between synchronous & Asynchronous counters. (Nov/Dec 2009, Nov 2017)

S.No	Asynchronous counters	Synchronous counters
1.	In this type of counter flip-flops are connected in such a way that output of 1 st flip-flop drives the clock for the next flip - flop.	In this type there is no connection between output of first flip-flop and clock input of the next flip – flop
2	All the flip-flops are not clocked simultaneously	All the flip-flops are clocked simultaneously
3	Logic circuit is very simple even for more number of states	Design involves complex logic circuit as number of states increases
4	Counters speed is low.	Counters speed is high.

6. State the Steps or Design procedure for Synchronous Counter.

Preparation of 1). State Diagram

- 2). State Table
- 3). State Assignment
- 4). Excitation Table (Consider which Memory Unit Using)
- 5). K-Map
- 6). Circuit Diagram

7. What is modulo-N counter?

A modulo-*n* counter will count *n* states. For example a mod-6 counter will count the sequence 000,001,010,011,100,101 and then recycles to 000. Mod -6 counter skips 110 and 111 states and it goes through only six different states.

DESIGN OF RIPPLE COUNTERS

3-Bit Asynchronous Binary Counter/ modulo -7 ripple counter:

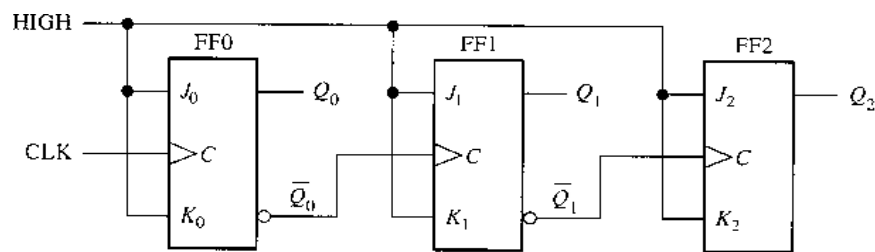
Design a 3-bit binary counter using T-flip flops. [NOV – 2019]

Explain about 3-Bit Asynchronous binary counter.

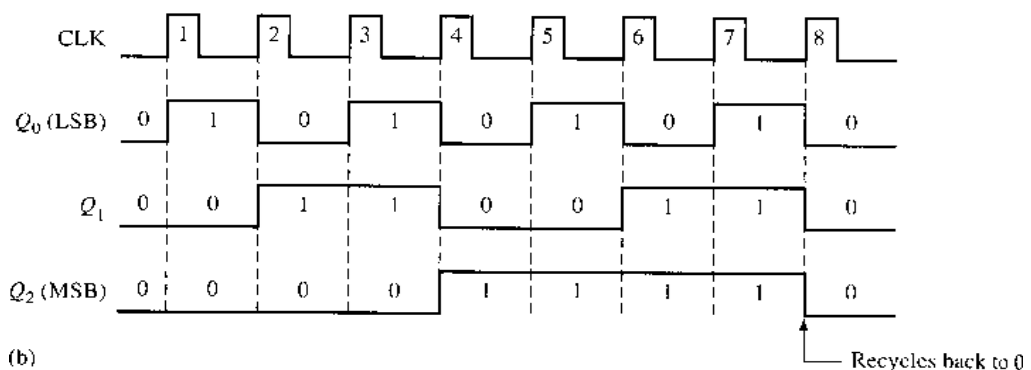
(Nov -2009)

The following is a three-bit asynchronous binary counter and its timing diagram for one cycle. It works exactly the same way as a two-bit asynchronous binary counter mentioned above, except it has eight states due to the third flip-flop.

Clock Pulse	Q_2	Q_1	Q_0
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0



(a)



(b)

Asynchronous counters are commonly referred to as ripple counters for the following reason: The effect of the input clock pulse is first “felt” by FF0. This effect cannot get to FF1 immediately because of the propagation delay through FF0. Then there is the propagation delay through FF1 before FF2 can be triggered. Thus, the effect of an input clock pulse “ripples” through the counter, taking some time, due to propagation delays, to reach the last flip-flop.

DESIGN OF SYNCHRONOUS COUNTERS

Design and analyze of clocked sequential circuit with an example.

The procedure for designing synchronous sequential circuit is given below,

1. *From the given specification, Draw the state diagram.*
2. *Plot the state table.*
3. *Reduce the number of states if possible.*
4. *Assign binary values to the states and plot the transition table by choosing the type of Flip-Flop.*
5. *Derive the Flip flop input equations and output equations by using K-map.*
6. *Draw the logic diagram.*

State Diagram:

- State diagram is the *graphical representation of the information available in a state table.*
- In state diagram, a state is represented by a circle and the transitions between states are indicated by directed lines connecting the circles.

State Table:

- A state table gives the time sequence of inputs, outputs and flip flops states. The table consists of four sections labeled present state, next state, input and output.
- The present state section shows the states of flip flops A and B at any given time 'n'. The input section gives a value of x for each possible present state.
- The next state section shows the states of flip flops one clock cycle later, at time n+1.

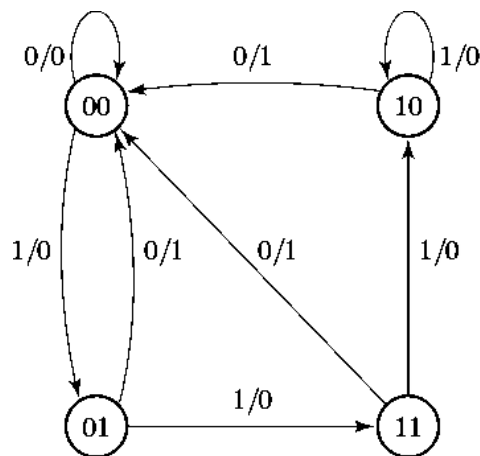
The state table for the circuit is shown. This is derived using state equations.

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

The above state table can also be expressed in different forms as follows.

Present State		Next State				Output	
		$x = 0$		$x = 1$		$x = 0$	$x = 1$
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

The state diagram for the logic circuit in below figure.



Flip-Flop Input Equations:

The part of the circuit that generates the inputs to flip flops is described algebraically by a set of Boolean functions called flip flop input equations.

The flip flop input equations for the circuit is given by,

$$D_A = Ax + Bx$$

$$D_B = Ax$$

TWO MARKS

1. Define state diagram.

State diagram is the graphical representation of the information available in a state table.

In state diagram, a state is represented by a circle and the transitions between states are indicated by directed lines connecting the circles.

2. What is the use of state diagram?

- i) Behavior of a state machine can be analyzed rapidly.
- ii) It can be used to design a machine from a set of specification.

3. What is state table? (Nov 2018)

A state table is a table that represents relationship between inputs, outputs and flip-flop states, is called state table. Generally it consists of four section present state, next state, input and output.

4. What is a state equation?

A state equation also called, as an application equation is an algebraic expression that specifies the condition for a flip-flop state transition. The left side of the equation denotes the next state of the flip-flop and the right side, a Boolean function specifies the present state.

5. Define sequential circuit.

Sequential circuits are circuits in which the output variables dependent not only on the present input variables but they also depend up on the past output of these input variables.

6. What do you mean by present state?

The information stored in the memory elements at any given time defines the present state of the sequential circuit.

7. What do you mean by next state?

The present state and the external inputs determine the outputs and the next state of the sequential circuit.

8. Define synchronous sequential circuit.

SynchronousSequential circuits are circuits in which the signals can affect the memory elements only at discrete instant of time.

9. What are the steps for the design of asynchronous sequential circuit?

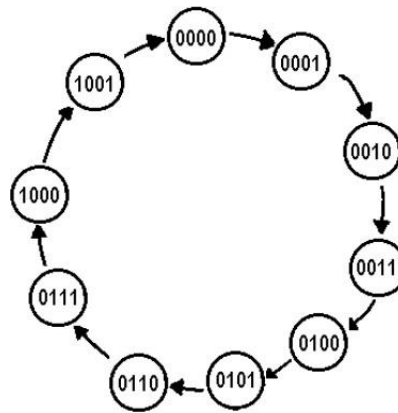
- i) Construction of primitive flow table
- ii) Reduction of flow table
- iii) State assignment is made
- iv) Realization of primitive flow table

Design of a Synchronous Decade Counter Using JK Flip- Flop (Apr 2018, Nov 2018)

A synchronous decade counter will count from zero to nine and repeat thesequence.

State diagram:

The state diagram of this counter is shown in Fig.



Excitation table:

Present State				Next State				Output							
Q ₃	Q ₂	Q ₁	Q ₀	Q ₃	Q ₂	Q ₁	Q ₀	J ₃	K ₃	J ₂	K ₂	J ₁	K ₁	J ₀	K ₀
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	0	0	0	0	X	1	0	X	0	X	X	1

K-Map:

		Q ₁ Q ₀			
		00	01	11	10
Q ₃ Q ₂	00	1	X	X	1
	01	1	X	X	1
	11	X	X	X	X
	10	1	X	X	X

J₀ = 1

		Q ₁ Q ₀			
		00	01	11	10
Q ₃ Q ₂	00	X	1	1	X
	01	X	1	1	X
	11	X	X	X	X
	10	X	1	X	X

K₀ = 1

	Q_1Q_0			
	00	01	11	10
Q_3Q_2				
00		1	X	X
01		1	X	X
11	X	X	X	X
10			X	X

$$J_1 = \bar{Q}_3 Q_0$$

	Q_1Q_0			
	00	01	11	10
Q_3Q_2				
00	X	X	1	
01	X	X	1	
11	X	X	X	X
10	X	X	X	X

$$K_1 = \bar{Q}_3 Q_0$$

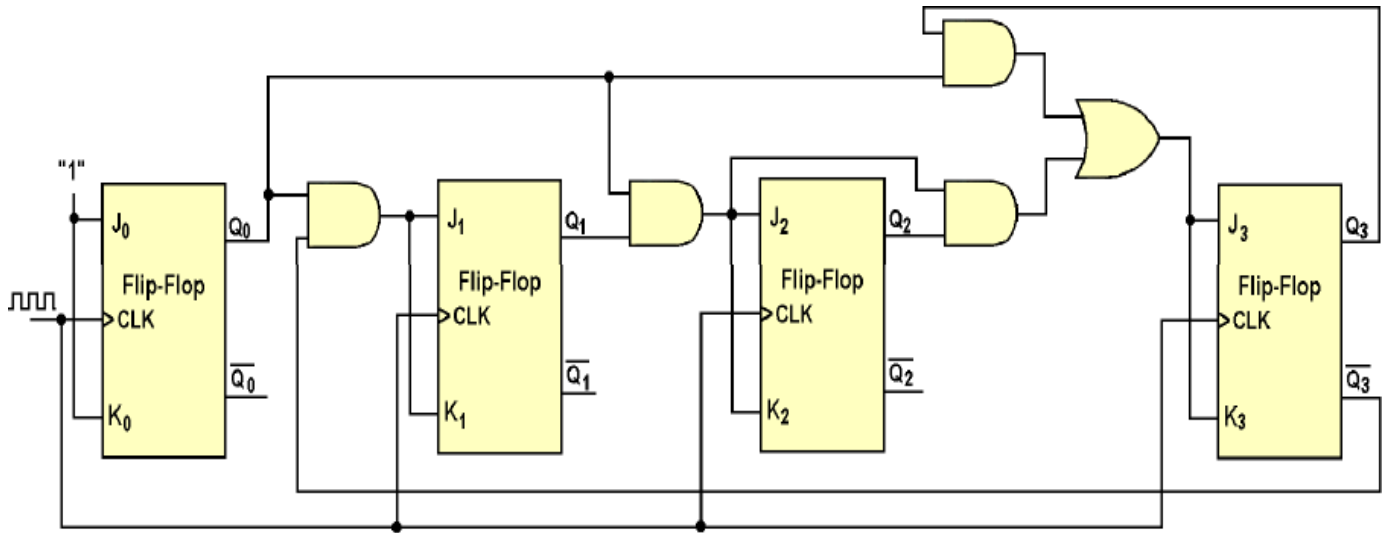
	Q_1Q_0			
	00	01	11	10
Q_3Q_2				
00				
01			1	
11	X	X	X	X
10	X	X	X	X

$$J_3 = Q_3 Q_0 + Q_2 Q_1 Q_0$$

	Q_1Q_0			
	00	01	11	10
Q_3Q_2				
00	X	X	X	X
01	X	X	X	X
11	X	X	X	X
10		1	X	X

$$K_3 = Q_3 Q_0 + Q_2 Q_1 Q_0$$

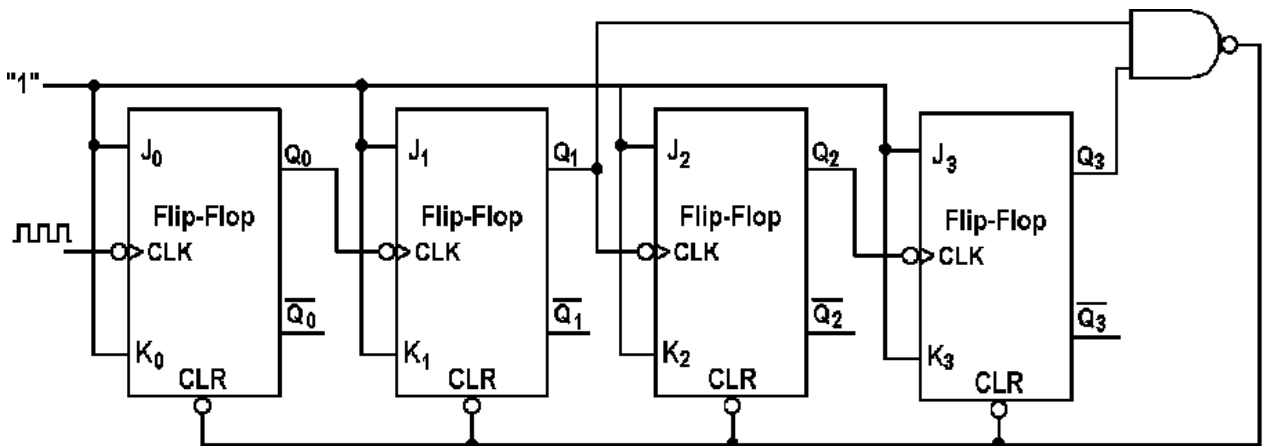
Logic Diagram:



Design of an Asynchronous Decade Counter Using JK Flip- Flop.

An asynchronous decade counter will count from zero to nine and repeat thesequence. Since the JK inputs are fed from the output of previous flip-flop,therefore, the design will not be as complicated as the synchronous version.

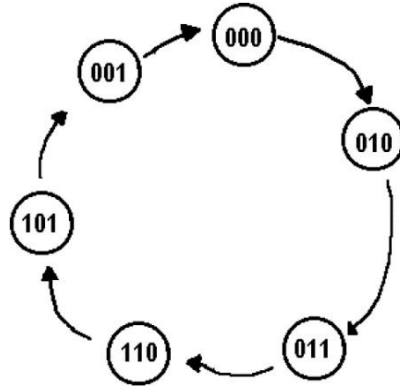
At the ninth count, the counter is reset to begin counting at zero. The NAND gateis used to reset the counter at the ninth count. At the ninth count the outputs offlip-flop Q3 and Q1 will be high simultaneously. This will cause the output ofNAND to go to logic “0” that would reset the flip-flop. The logic design of thecounter is shown in Fig.



Design of a Synchronous Modulus-Six Counter Using SR Flip-Flop(Nov 2017)

The modulus six counters will count 0, 2, 3, 6, 5, and 1 and repeat the sequence. This modulus six counter requires three SR flip-flops for the design.

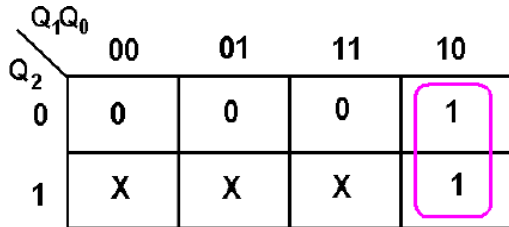
State diagram:



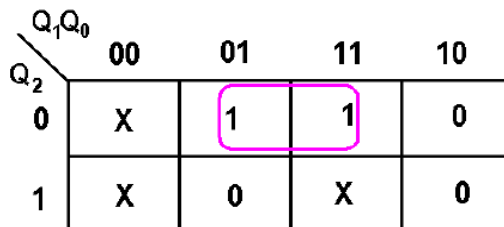
Truth table:

[Present State			Next State			Output					
Q ₂	Q ₁	Q ₀	Q ₂	Q ₁	Q ₀	R ₂	S ₂	R ₁	S ₁	R ₀	S ₀
0	0	0	0	1	0	0	X	1	0	0	X
0	1	0	0	1	1	0	X	X	0	1	0
0	1	1	1	1	0	1	0	X	0	0	1
1	1	0	1	0	1	X	0	0	1	1	0
1	0	1	0	0	1	0	1	0	X	X	0
0	0	1	0	0	0	0	X	0	X	0	1

K-Map:



$$R_0 = Q_1 \cdot \bar{Q}_0$$



$$S_0 = \bar{Q}_2 \cdot Q_0$$

	Q_1Q_0		00	01	11	10
Q_2	0	1	1	0	X	X
	1	0	X	0	X	0

$$R_1 = \bar{Q}_1 \cdot \bar{Q}_0$$

	Q_1Q_0		00	01	11	10
Q_2	0	1	0	X	0	0
	1	0	X	X	X	1

$$S_1 = Q_2$$

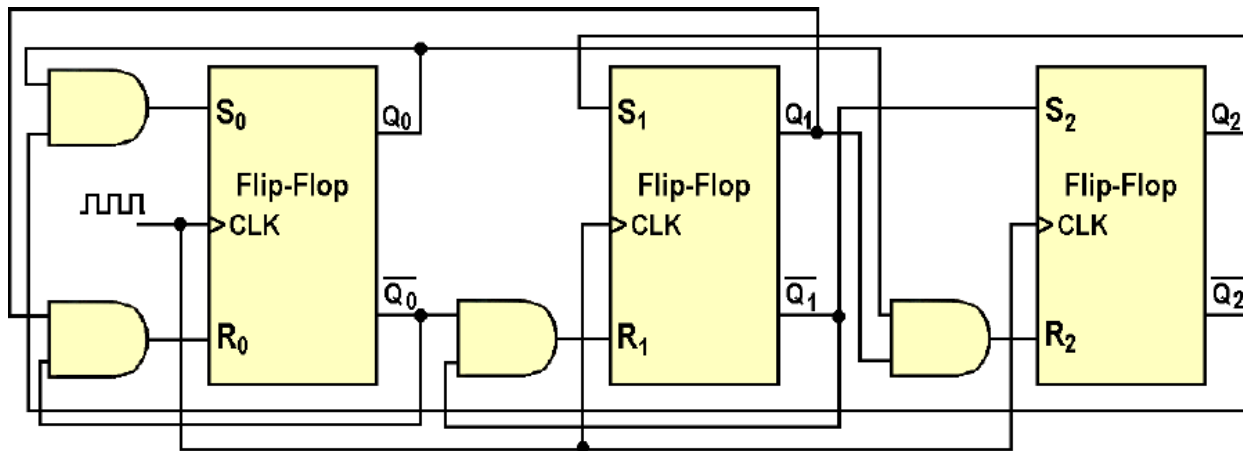
	Q_1Q_0		00	01	11	10
Q_2	0	1	0	0	1	0
	1	0	X	0	X	X

$$R_2 = Q_1 \cdot Q_0$$

	Q_1Q_0		00	01	11	10
Q_2	0	1	X	X	0	X
	1	0	X	1	X	0

$$S_2 = \bar{Q}_1$$

Logic Diagram:



SHIFT REGISTERS

Explain various types of shift registers. (or) Explain the operation of a 4-bit bidirectional shift register.

(Or) What are registers? Construct a 4 bit register using D-flip flops and explain the operations on the register. (or) With diagram explain how two binary numbers are added serially using shift registers.

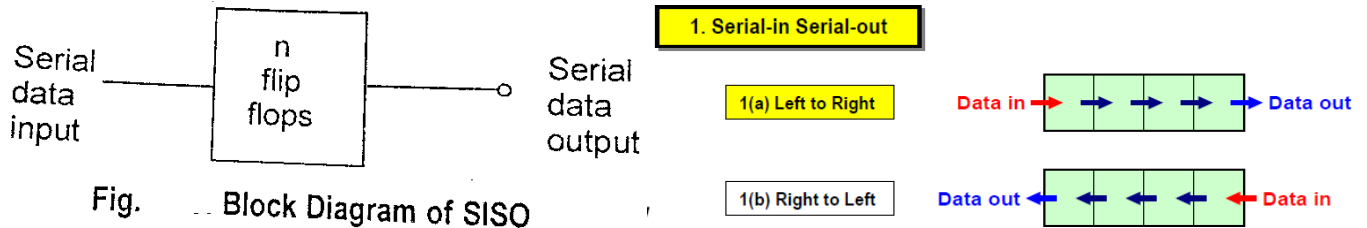
(Apr – 2019)[NOV – 2019]

- A register is simply a group of Flip-Flops that can be used to store a binary number.
- There must be one Flip-Flop for each bit in the binary number.
- For instance, a register used to store an 8-bit binary number must have 8 Flip-Flops.
- The Flip-Flops must be connected such that the binary number can be entered (shifted) into the register and possibly shifted out.
- A group of Flip-Flops connected to provide either or both of these functions is called a **shift register**.
- A register capable of shifting the binary information held in each cell to its neighboring cell in a selected direction is called a shift register.

- There are four types of shift registers namely:
 1. Serial In Serial Out Shift Register,
 2. Serial In Parallel Out Shift Register
 3. Parallel In Serial Out Shift Register
 4. Parallel In Parallel Out Shift Register

1. Serial In Serial Out Shift Register

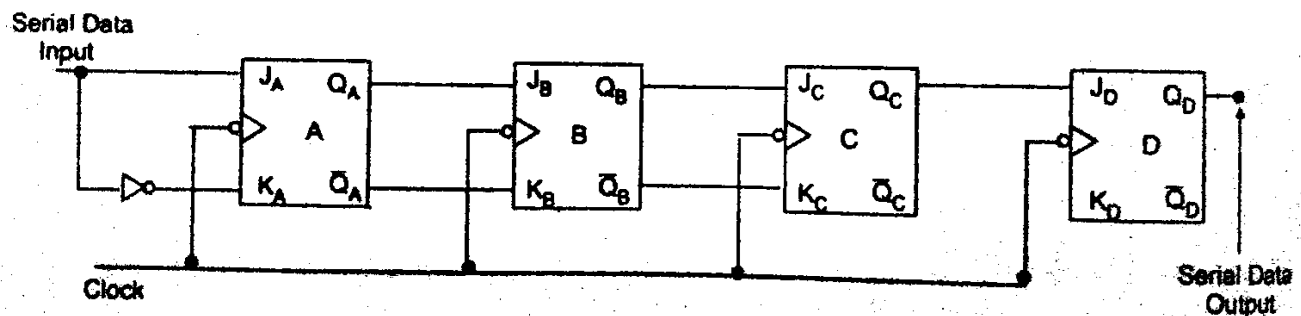
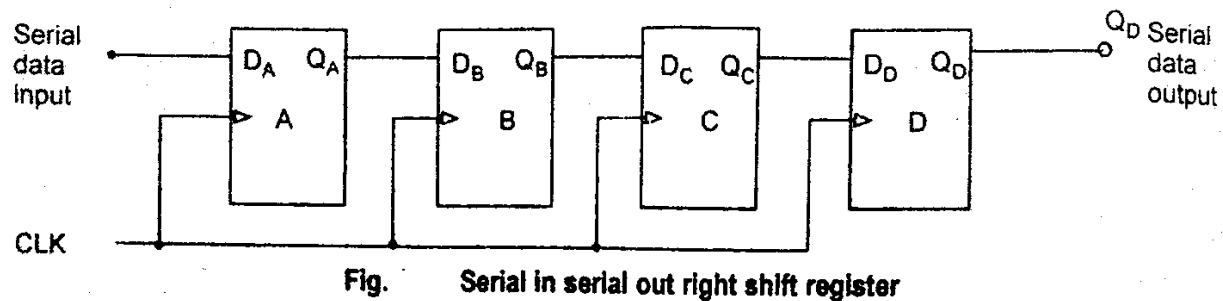
- The block diagram of a serial out shift register is as below.



- As seen, it accepts data serially .i.e., one bit at a time on a single input line. It produces the stored information on its single output also in serial form.
- Data may be shifted left using shift left register or shifted right using shift right register.

Shift Right Register

The circuit diagram using D flip-flops is shown in figure



- As shown in above figure, the clock pulse is applied to all the flip-flops simultaneously.
- The output of each flip-flop is connected to D input of the flip-flop at its right.
- Each clock pulse shifts the contents of the register one bit position to the right.
- New data is entered into stage A whereas the data presented in stage D are shifted out.

- For example, consider that all stages are reset and a steady logical 1 is applied to the serial input line.
- When the *first clock pulse* is applied, flip-flop A is set and all other flip-flops are reset.
- When the *second clock pulse* is applied, the '1' on the data input is shifted into flip-flop A and '1' that was in flip flop A is shifted to flip-flop B.
- This continues till all flip-flop sets.
- The data in each stage after each clock pulse is shown in table below

Shift Pulse	Serial Data Input	Q _A	Q _B	Q _C	Serial Output Q _D
0	1	0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	1	1	1	1	1

Shift Left Register

The figure below shows the shift left register using D flip-flops.

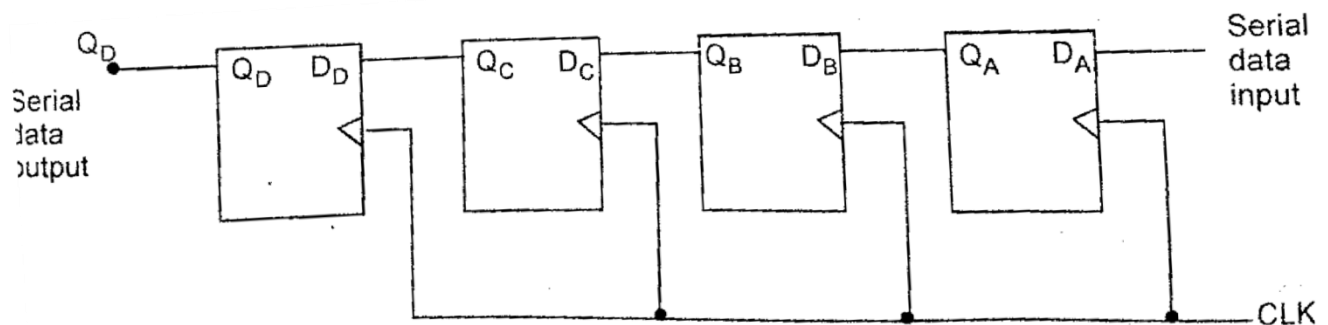


Fig. : Serial in serial out shift left register

- The clock is applied to all the flip-flops simultaneously. The output of each flip-flop is connected to D input of the flip-flop at its left.
- Each clock pulse shifts the contents of the register one bit position to the left.
- Let us illustrate the entry of the 4-bit binary number 1111 into the register beginning with the right most bit.
- When the *first clock pulse* is applied, flip flop A is set and all other flip-flops are reset.
- When *second clock pulse* is applied, '1' on the data input is shifted into flip-flop A and '1' that was in flip flop A is shifted to flip-flop B. This continues till all flip-flop are set.
- The data in each stage after each clock pulse is shown in table below.

Q_D	Q_C	Q_B	Q_A	Serial Input Data	Clock Pulse
0	0	0	0	1	0
0	0	0	1	1	1
0	0	1	1	1	2
0	1	1	1	1	3
1	1	1	1	1	4

2. Serial in Parallel out shift register:

A 4 bit serial in parallel out shift register is shown in figure.

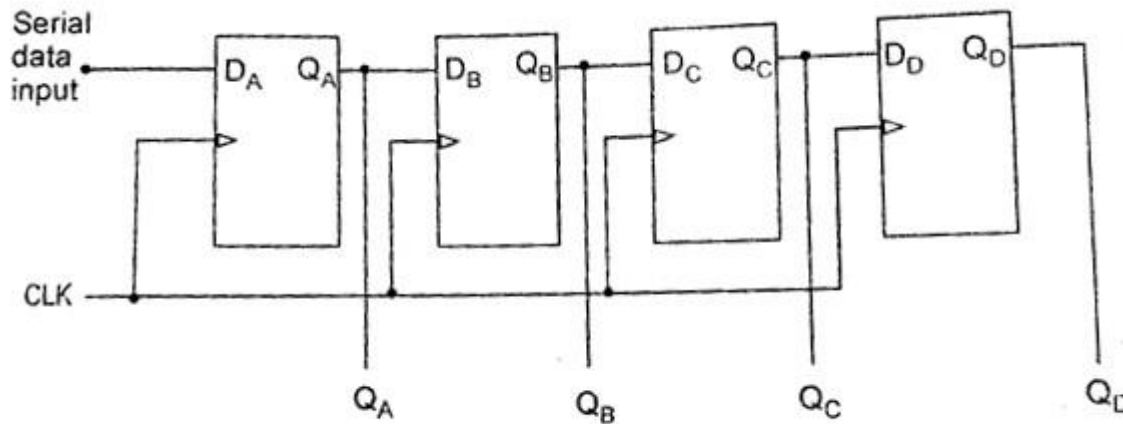
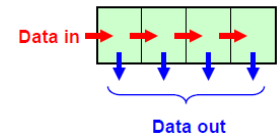


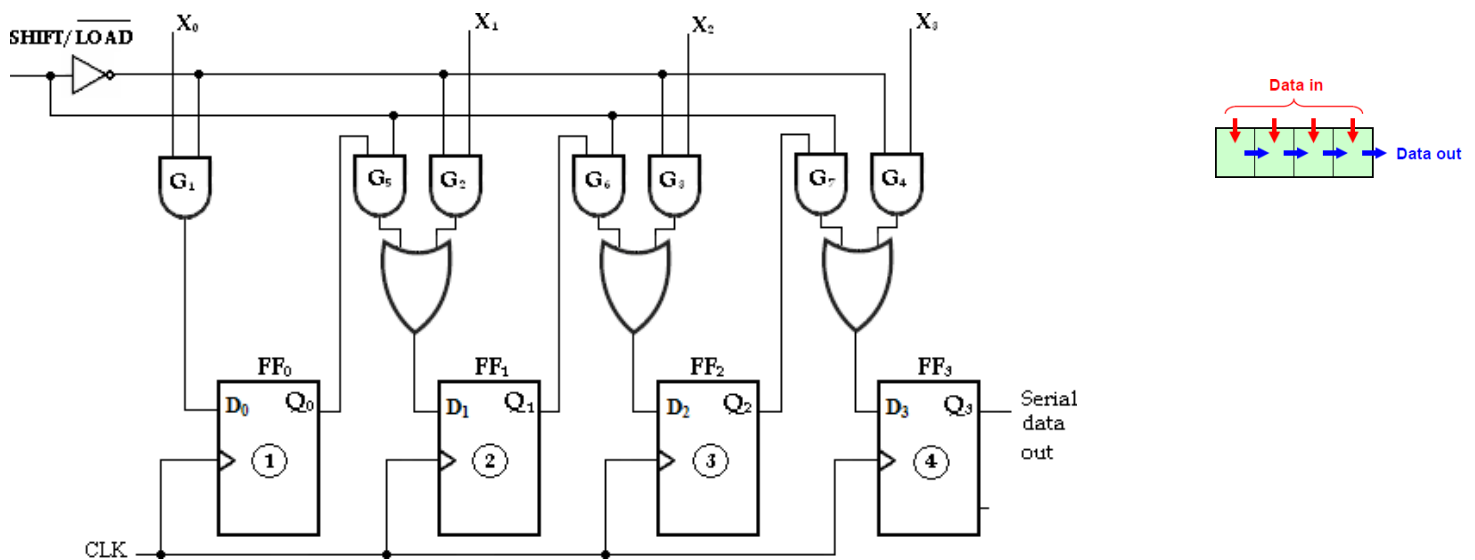
Fig. 3.42: Serial in parallel out shift register

- It consists of one serial input and outputs are taken from all the flip-flops simultaneously.
- The output of each flip-flop is connected to D input of the flip-flop at its right. Each clock pulse shifts the contents of the register one bit position to the right.
- For example, consider that all stages are reset and a steady logical '1' is applied to the serial input line.
- When the *first clock pulse* is applied flip flop A is set and all other flip-flops are reset.
- When the *second pulse* is applied the '1' on the data input is shifted into flip flop A and '1' that was in flip flop A is shifted into flip-flop B. This continues till all flip-flops are set. The data in each stage after each clock pulse is shown in table below.

Shift Pulse	Serial Data Input	Parallel Outputs			
		Q _A	Q _B	Q _C	Q _D
0	1	0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	1	1	1	1	1

3. Parallel In Serial Out Shift register:

- For register with parallel data inputs, register the bits are entered simultaneously into their respective stages on parallel lines.
- A four bit parallel in serial out shift register is shown in figure. Let A,B,C and D be the four parallel data input lines and $\overline{\text{SHIFT/LOAD}}$ is a control input that allows the four bits of data to be entered in parallel or shift the serially.



- When $\overline{\text{SHIFT/LOAD}}$ is low, gates G1 through G3 are enabled, allowing the data at parallel inputs to the D input of its respective flip-flop. When the clock pulse is applied the flip-flops with D=1 will set and those with D=0 will reset, thereby storing all four bits simultaneously.
- When $\overline{\text{SHIFT/LOAD}}$ is high, AND gates G1 through G3 are disabled and gates G4 through G6 are enabled, allowing the data bits to shift right from one stage to next. The OR gates allow either the normal shifting operation or the parallel data entry operation, depending on which AND gates are enabled by the level on the $\overline{\text{SHIFT/LOAD}}$ input.

Parallel In Parallel Out Shift Register:

- In parallel in parallel out shift register, data inputs can be shifted either in or out of the register in parallel.
- A four bit parallel in parallel out shift register is shown in figure. Let A, B, C, D be the four parallel data input lines and Q_A, Q_B, Q_C and Q_D be four parallel data output lines. The $\overline{SHIFT/LOAD}$ is the control input that allows the four bits data to enter in parallel or shift the serially.

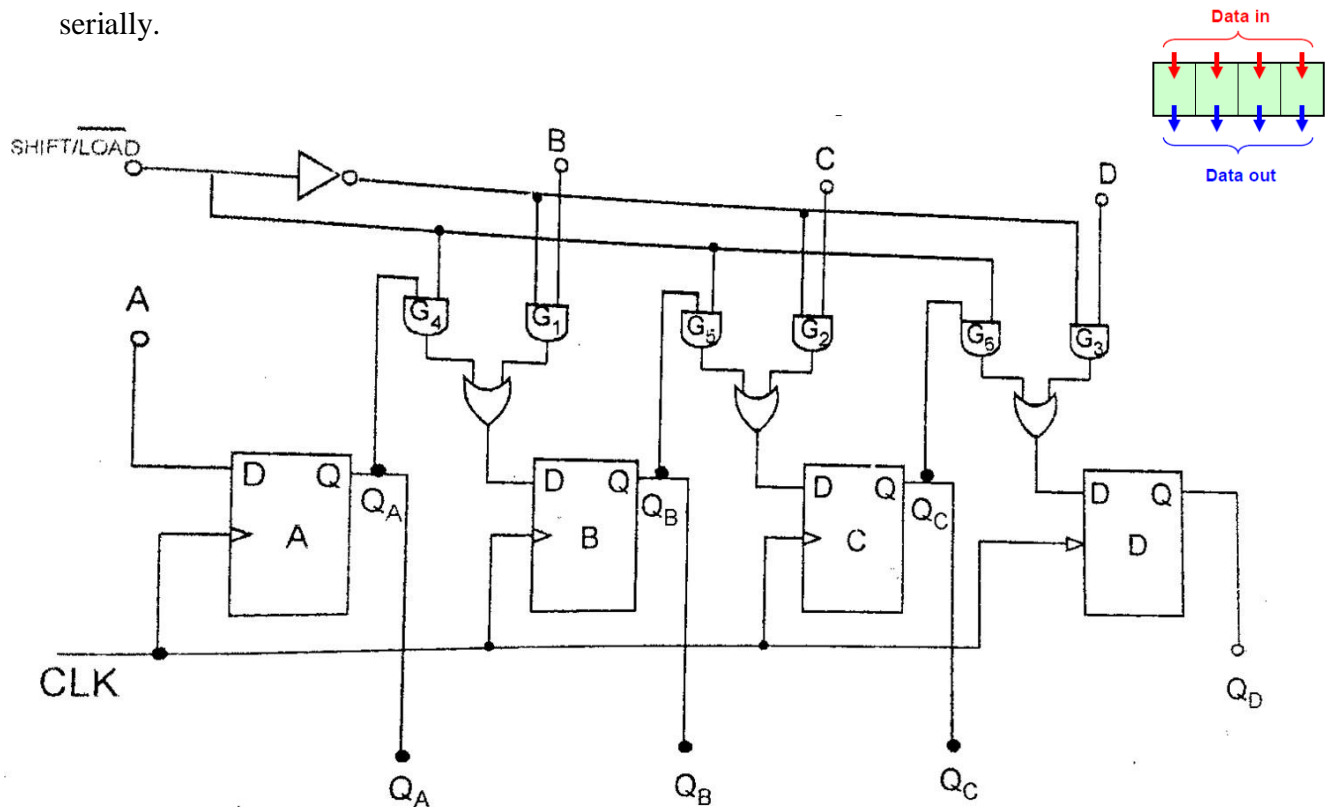


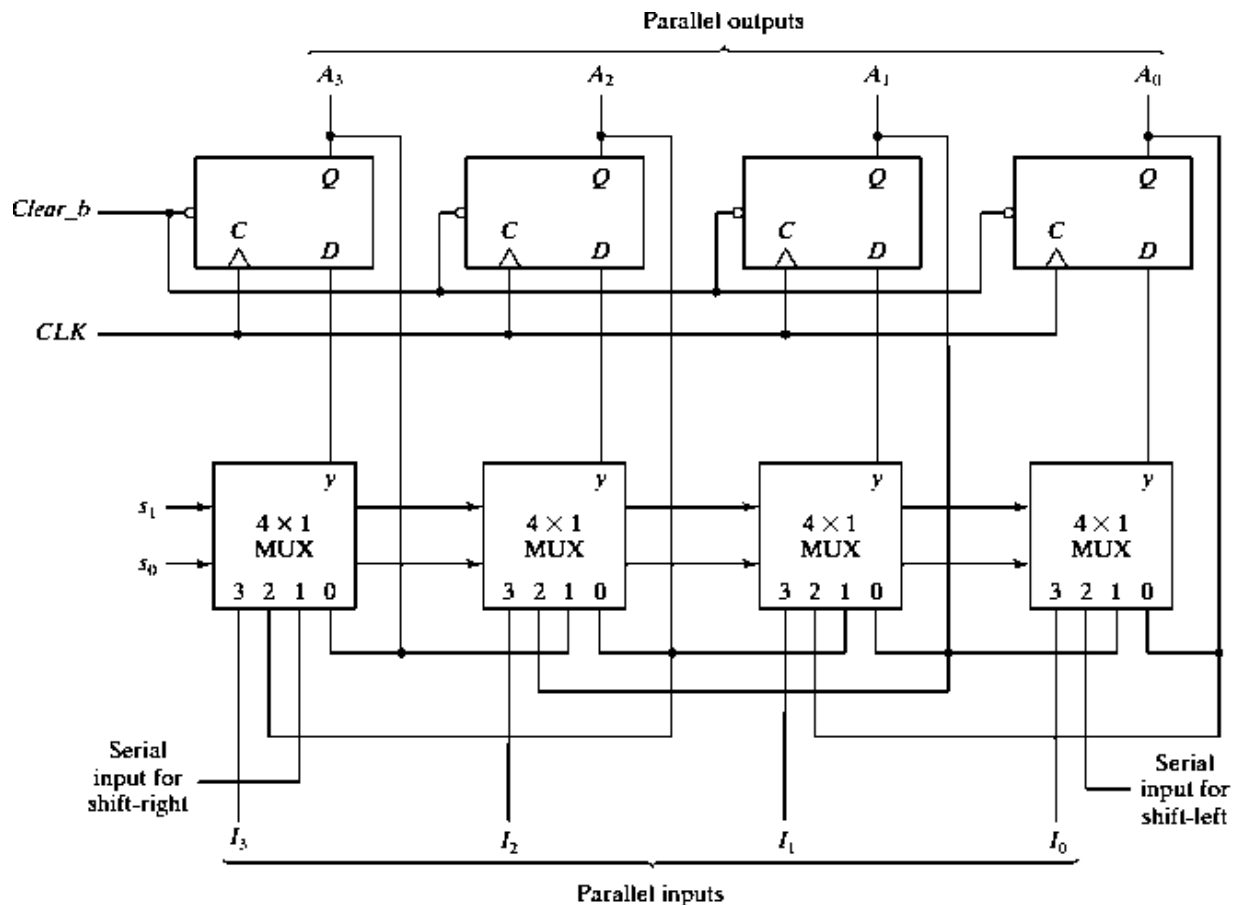
Fig. : Parallel in parallel out shift register

- When $\overline{SHIFT/LOAD}$ is low, gates G1 through G3 are enabled, allowing the data at parallel inputs to the D input of its respective flip-flop. When the clock pulse is applied, the flip-flops with D = 1 will set those with D = 0 will reset thereby storing all four bits simultaneously. These are immediately available at the outputs Q_A, Q_B, Q_C and Q_D .
- When $\overline{SHIFT/LOAD}$ is high, gates G1, through G3 are disabled and gates G4 through G6 are enabled allowing the data bits to shift right from one stage to another. The OR gates allow either the normal shifting operation or the parallel data entry operation, depending on which AND gates are enabled by the level on the $\overline{SHIFT/LOAD}$ input.

Universal Shift Register:

Explain about universal shift register. (Apr -2018)

- A register that can shift data to right and left and also has parallel load capabilities is called universal shift register.
- It has the following capabilities.
 1. A clear control to clear the register to 0.
 2. A clock input to synchronize the operations.
 3. A shift right control to enable the shift right operation and the associated serial input and output lines.
 4. A shift left control to enable the shift left operation and the associated serial input and output lines.
 5. A parallel load control to enable a parallel transfer and the n input lines.
 6. n parallel output lines.
 7. A control state that leaves the information in the register unchanged in the presence of the clock.



- The diagram of 4-bit universal shift register that has all that capabilities listed above is shown in figure. It consists of four D flip-flop and four multiplexers. All the multiplexers have two common selection inputs S_1 and S_0 . Input 0 is selected when $S_1S_0=00$, input 1 is selected when $S_1S_0=01$ and similarly for other two inputs.
- The selection inputs control the mode of operation of the register. When $S_1S_0=00$, the present value of the register is applied to the D inputs of the flip-flop. The next clock pulse transfers into each flip-flop the binary value it held previously, and no change of state occurs.
- When $S_1S_0=01$, terminal 1 of the multiplexer inputs has a path to be the D inputs of the flip-flops. This causes a shift right operation, with the serial input transferred into flip-flop A_3 .
- When $S_1S_0=10$, a shift left operation results with the other serial input going into flip-flop A_0 . Finally, when $S_1S_0 = 11$, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock edge. The function table is shown below.

Mode Control		
s_1	s_0	Register Operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

SHIFT REGISTER COUNTERS:

Explain about Johnson and Ring counter. (Nov 2018)

Most common shift register counters are Johnson counter and ring counter.

Johnson counter:

- A 4 bit Johnson counter using D flip-flop is shown in figure. It is also called shift counter or twisted counter.

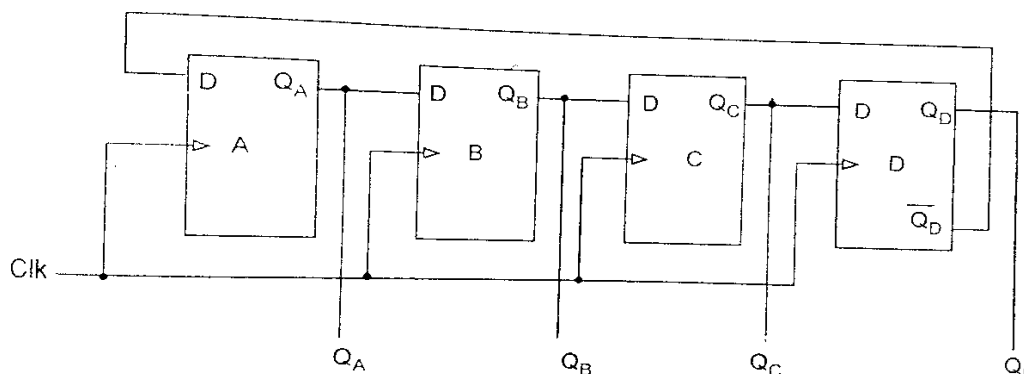


Fig. : Johnson Counter

- The output of each flip-flop is connected to D input of the next stage. The inverted output of last flip-flop $\overline{Q_D}$ is connected to the D input of the first flip-flop A.
- Initially, assume that the counter is reset to 0. i.e., $Q_A Q_B Q_C Q_D = 0000$. The value at $D_B = D_C = D_D = 0$, whereas $D_A = 1$ since $\overline{Q_D}$.
- When the *first clock pulse* is applied, the first flip-flop A is set and the other flip-flops are reset. i.e., $Q_A Q_B Q_C Q_D = 1000$.
- When the *second clock pulse* is applied, the counter is $Q_A Q_B Q_C Q_D = 1100$. This continues and the counter will fill up with 1's from left to right and then it will fill up with 0's again.
- The sequence of states is shown in the table. As observed from the table, a 4-bit shift counter has 8 states. In general, an n -flip-flop Johnson counter will result in $2n$ states.

Clock Pulse	Q_A	Q_B	Q_C	Q_D	$\overline{Q_D}$
0	0	0	0	0	1
1	1	0	0	0	1
2	1	1	0	0	1
3	1	1	1	0	1
4	1	1	1	1	0
5	0	1	1	1	0
6	0	0	1	1	0
7	0	0	0	1	0
0	0	0	0	0	1

The timing diagram of Johnson counter is as follows:

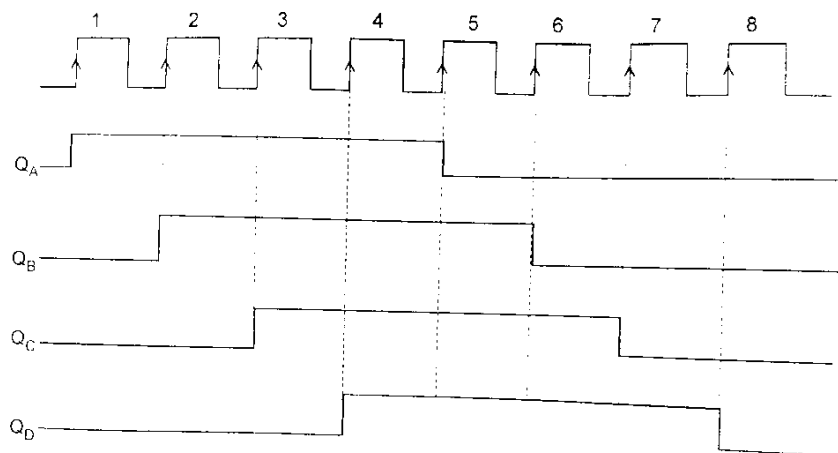


Fig. . : Timing Diagram of Johnson Counter

Ring Counter:

A 4-bit ring counter using D Flip-Flop is shown in figure.

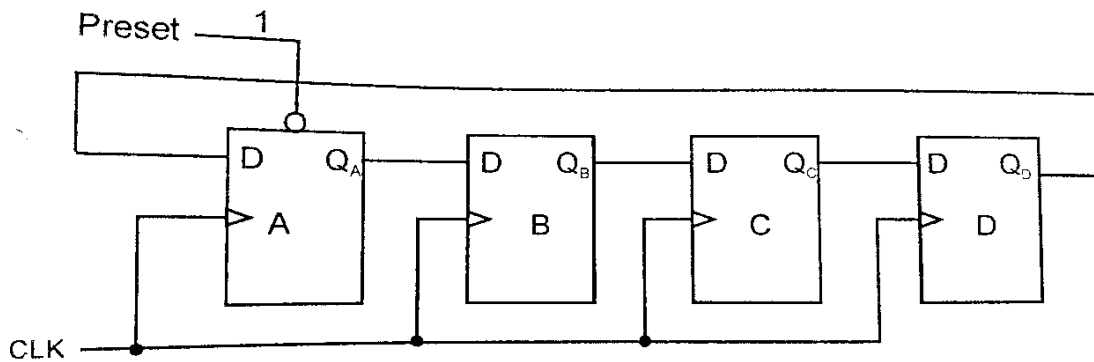


Fig. : Ring Counter

- As shown in figure, the true output of flip-flop D. i.e., Q_D is connected back to serial input of flip-flop A.
- Initially, 1 preset into the first flip-flop and the rest of the flip-flops are cleared i.e., $Q_A Q_B Q_C Q_D = 1000$.
- When the *first clock pulse is applied*, the second flip-flop is set to 1 while the other three flip-flops are reset to 0.
- When the second clock pulse is applied, the '1' in the second flip-flop is shifted to the third flip-flop and so on.
- The truth table which describes the operation of the ring counter is shown below.

Clock Pulse	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
0	1	0	0	0

- As seen a 4-bit ring counter has 4 states. In general, an n -bit ring counter has n states. Since a single '1' in the register is made to circulate around the register, it is *called a ring counter*. The timing diagram of the ring counter is shown in figure.

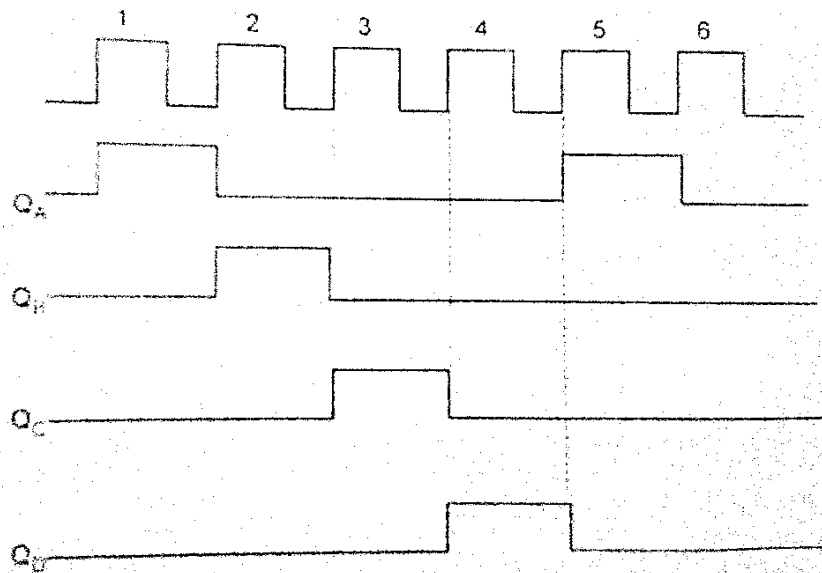


Fig. Timing Diagram of Ring Counter

TWO MARKS

1. Define registers.

A register is a group of flip-flops. An-bit register has a group of n flip-flops and is capable of storing any binary information/number containing n-bits.

2. Define shift registers.

A register capable of shifting its binary information in one or both directions is called as a shift register. It consists of a chain of flip flops in cascade, with the output of one flip flop connected to the input of the next flip-flop

3. What are the different types of shift registers?[Nov 2010, April 2007, Apr 2018, Nov 2018]

- ✓ Serial In Serial Out Shift Register
- ✓ Serial In Parallel Out Shift Register
- ✓ Parallel In Serial Out Shift Register
- ✓ Parallel In Parallel Out Shift Register
- ✓ Bidirectional Shift Register

4. State the applications of shift register.

Shift registers are widely used in

- ✓ Time delay circuits
- ✓ As Serial to parallel converter
- ✓ As Parallel to serial converters
- ✓ As Counters

5. Define Shift Register Counter.

A shift register can also be used as a counter. A shift register with the serial output connection back to the serial input is called Shift register counter

6. What is bi-directional shift register and unidirectional shift register?

A register capable of shifting both right and left is called bi-directional shift register. A register capable of shifting only one direction is called unidirectional shift register.

7. What are the two types of shift register counters?[April/May 2007,Nov/Dec 2006,2011,2012]

There are 2 types of shift Register counters are:

Ring counter:

A ring counter is a circular shift register with only one flip flop being set, at any particular time, all others are cleared.

Johnson counters:

The Johnson counter is K-bit switch-tail rings counter $2k$ decoding gates to provide outputs for $2k$ timing signals.

8. How can a SIPO shift register is converted in to SISO shift register? (Apr/May 2010)

By taking output only on the Q output of last flip flop SIPO shift register is converted in to SISO shift register.

9. What is bi-directional shift register and unidirectional shift register?

A register capable of shifting both right and left is called bi-directional shift register. A register capable of shifting only one direction is called unidirectional shift register.

10. What is sequence generator?

The sequential circuit used to repeat a particular sequence repeatedly is called Sequence generator.

Write coding in HDL for various flip-flops.

D Flip Flops

```
module DFF (q, d, clock, reset);
input clock, reset, d;
output q;
reg q;
always @(posedge clock, negedge reset)
begin
    if (~ reset)
        q <= 1'b0;
    else
        q <= d;
    end
end module
```

T Flip-Flop

```
module TFF (q, t, clock, reset);
input clock, reset, t;
output q;
reg q;
always @(posedge clock, negedge reset)
begin
    if (~ reset)  // same as if (reset==0)
        q <= 1'b0;
    else if (t)
        q <= ~q;
    end
end module
```

J-K Flip Flop

```
module JKFF (q, j, k, clock, reset);
input clock, reset, j, k;
output q;
```

```

reg q;
always @(posedge clock, negedge reset)
begin
    if (~ reset)
        q <= 1'b0;
    else
        begin
            case ({j, k})
                2'b00 : q <= q;
                2'b01 : q <= 0;
                2'b10 : q <= 1;
                2'b11 : q <= ~q;
            end case
        end
    end
end module.

```

T flip flop from D flip flop and gates

```

module T_FF (Q, T, CLK, RST);
    output Q;
    input T, CLK, RST;
    wire DT;
    assign DT = Q ^ T // T flip flop characteristic equation is  $Q(t+1) = Q \oplus T$ .
    DFF TF1 (Q, DT, CLK, RST); //Instantiate D flip flop.
endmodule.

```

JK flip flop from D flip flop and gates

```

module JK_FF (Q, J, K, CLK, RST);
    wire JK;
    assign JK = (J&~Q) | (~K & Q); // JK flip flop characteristic equation is  $Q(t+1) = J\bar{Q} + \bar{K}Q$ 
    //Instantiate D flip flop
    DFF JK1 (Q, JK, CLK, RST);
endmodule

```


Ripple Counter using T flip flop

```
module ripple counter (q, t, CLK, reset);  
input t, CLK, reset ;  
output [3:0] q;  
// Instantiate t flip flop  
    TFF t1 (q[0], t, CLK, reset);  
    TFF t2 (q[1], t, q[0], reset);  
    TFF t3 (q[2], t, q[1], reset);  
    TFF t4 (q[3], t, q[2], reset);  
end module
```

Synchronous Counter

```
module synchronous counter (CLK, reset, q);  
input CLK, reset;  
output [3:0] q;  
reg [3:0] q;  
always @ (posedge reset, posedge CLK)  
    begin  
        if (reset)  
            q <= 4'b0000;  
        else  
            q <= q + 1'b1;  
        end  
    end  
end module
```

Ripple Counter using D-flip flop

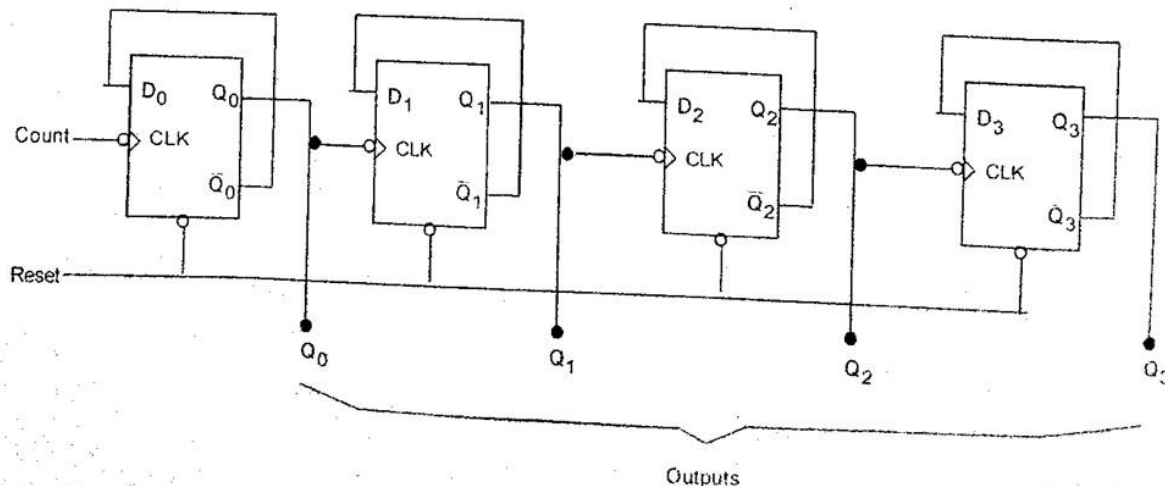


Fig. 3.50 Ripple counter using D-flipflop

```

module ripple_counter (Q3, Q2, Q1, Q0, count, reset);
    output Q3, Q2, Q1, Q0 ;
    input count, reset;
    //instantiate complementing flip flop.
    comp_DFF F0 (Q0, count, reset);
    comp_DFF F1 (Q1, Q0, reset);
    comp_DFF F2 (Q2, Q1, reset);
    comp_DFF F3 (Q3, Q2, reset);
endmodule.

//complementing D-flip flop
module comp_DFF (Q, CLK, reset);
    output Q;
    input CLK, RESET;
    reg Q;
    always @(negedge CLK, posedge Reset)
    if (~Reset), Q <= 1'b0;
    else Q <= #2 ~Q           //intra-assignment delay
endmodule

```

Universal Shift Register

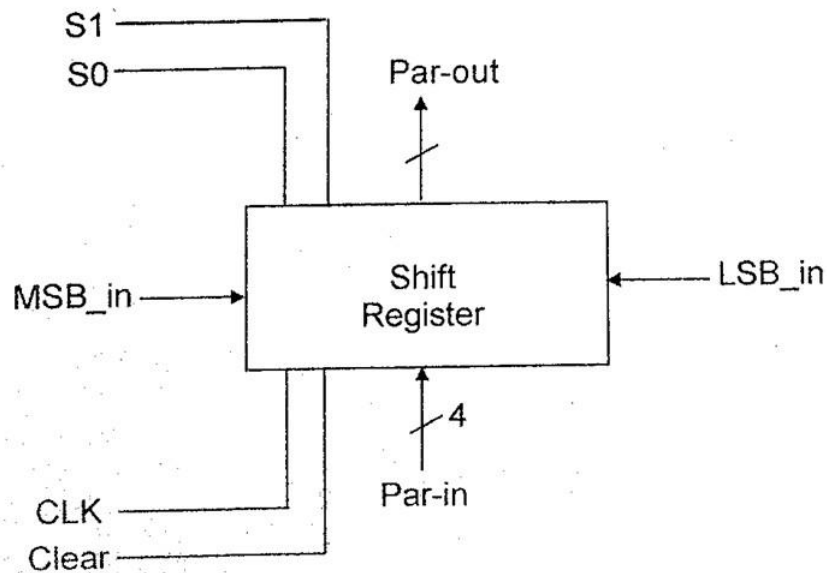


Fig. 3.51 Four Bit Universal Shift Register

Function Table		
Mode Control		
S_1	S_0	Operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

```

module shift register (S1, S0, LSB_in, MSB_in, Par_in, CLK, Clear, par_out);
input S1, S0, LSB_in, MSB_in, Clk, Clear;
input [3:0] par_in;
output [3:0] par_out;
reg [3:0] par_out;
always @ (posedge CLK, negedge Clear)
begin
    if (~ clear)
        par_out ← 4'b0000;
    else
        case ({S1, S0})
            2'b00 : par_out ← par_out;
            2'b01 : par_out ← {MSB_in, par_out [3:1]};
            2'b10 : par_out ← {par_out [2:0], LSB_in};
            2'b11 : par_out ← par_in;
        endcase
    end
endmodule.

```

Test Bench:

D flip flop

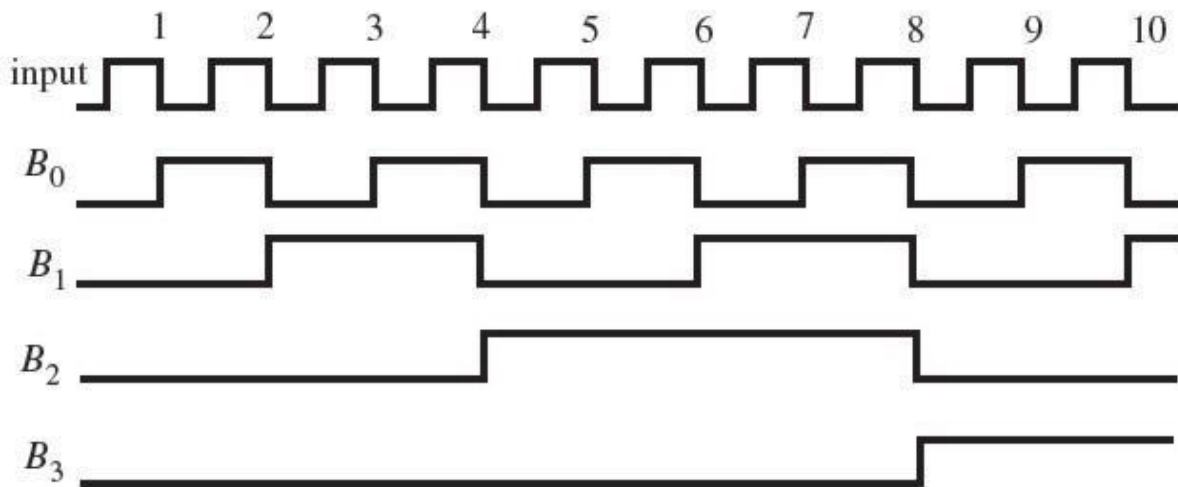
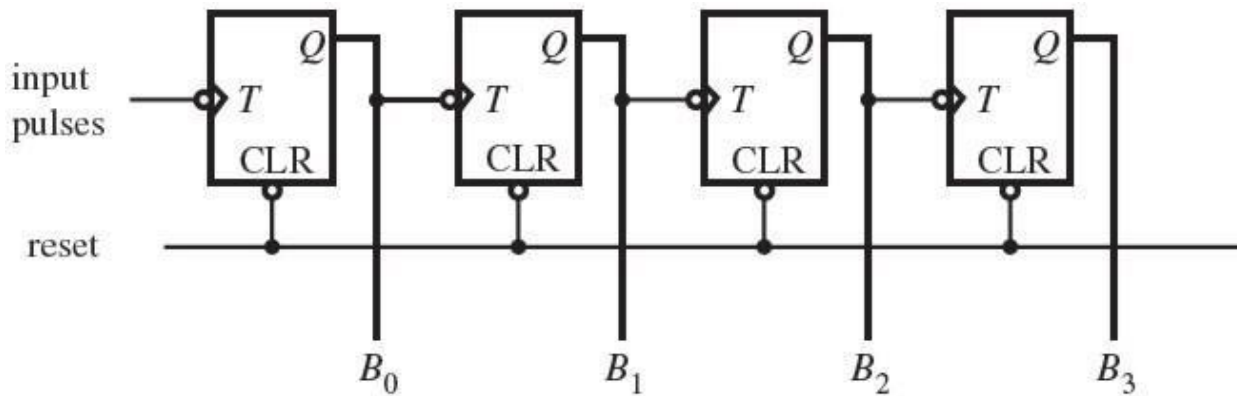
```
module DFF_test bench;
    wire tq;
    reg tclock, treset, td;
    DFF d1 (tclock, treset, td, tq);           //Instantiate D-flip flop module
    initial begin
        td = 0;
        tclock = 0;
        treset = 0;
        #3 treset = 1;
    end
    always #3 tclock = ~tclock;
    always #5 d = ~d
    initial #100 $ stop;
endmodule.
```

Synchronous Counter

```
module synchronouscounter_test;
    wire [3:0]tq;
    reg tCLK, treset;
    synchronous counter SC1 (tclk, treset, tq); //Instantiate synchronous counter module
    initial begin
        tclk = 0;
        treset = 0;
        #5 treset = 1;
        #5 treset = 0;
    end
    always #5 tCLK = ~tCLK
    initial #200 $stop
endmodule
```

Write the VHDL Code for 4-Bit Binary Up Counter and explain. (Apr 2019)
VHDL Code for 4-Bit Binary Up Counter

The clock inputs of all the flip-flops are connected together and are triggered by the input pulses. Thus, all the flip-flops change state simultaneously (in parallel).



```

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity vhdl_binary_counter is
port(C, CLR : in std_logic;
Q : out std_logic_vector(3 downto 0));
end vhdl_binary_counter;
architecture bhv of vhdl_binary_counter is
signal tmp: std_logic_vector(3 downto 0);
begin
process (C, CLR)
begin
if (CLR='1') then
tmp <= "0000";
elsif (C'event and C='1') then
tmp <= tmp + 1;
end if;
end process;
end architecture bhv;

```

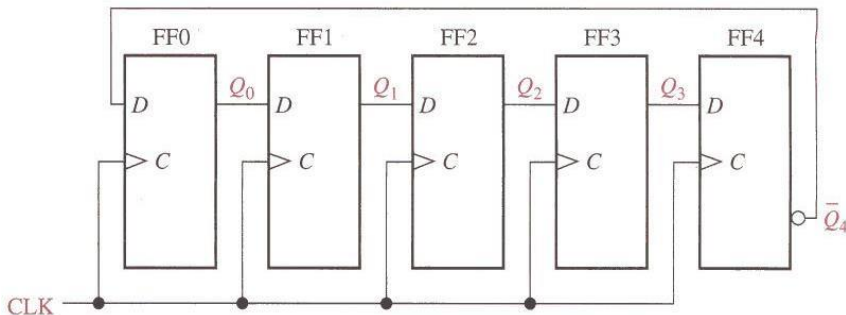
```

end if;
end process;
Q <= tmp;
end bhv;

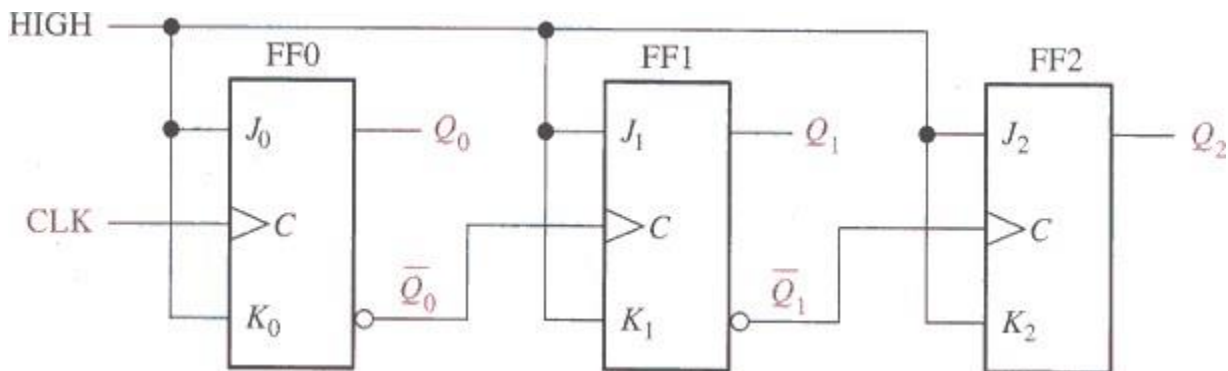
```

TWO MARKS

1. Draw 5-bit Johnson counter.



2. Draw the diagram of 3-bit ripple counter.



3. Give few applications of shift register.

- ✓ Serial to parallel converter
- ✓ Parallel to serial converter
- ✓ As a counter
- ✓ To introduce delay in a digital circuit.

UNIT III COMPUTER FUNDAMENTALS

Functional Units of a Digital Computer: Von Neumann Architecture – Operation and Operands of Computer Hardware Instruction – Instruction Set Architecture (ISA): Memory Location, Address and Operation – Instruction and Instruction Sequencing – Addressing Modes, Encoding of Machine Instruction – Interaction between Assembly and High Level Language.

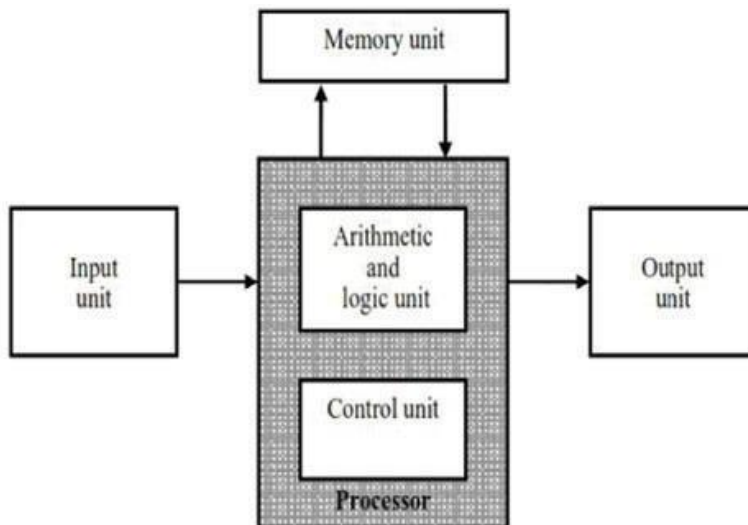
1. FUNCTIONAL UNITS OF A DIGITAL COMPUTER

1. Explain in detail about the components of a computer system. (12 or 16)
(Nov/Dec2014)(Nov/Dec2015) (May/June2016) (Nov/Dec 2016) Apr/ May 2018

Components of a computer system:

The five basic components of computer systems are,

- Input unit
 - Output unit
 - Arithmetic and logic unit
 - Memory unit
 - Control unit
- These units are interconnected by electrical cables to permit communication between them. This allows the computer to function as a system.



Input Unit:

- A computer must receive both data and program statements to function properly and be able to solve problems. The method of feeding data and programs to a computer is accomplished by an input device.

- Computer input devices read data from a source, such as magnetic disks, and translate that data into electronic impulses for transfer into the CPU. Some typical input devices are a keyboard, a mouse or a scanner.

Output Unit

- The output unit is the counterpart of the input unit. Its function is to send processed results to the outside world.
- The most familiar example of such a device is a printer. Printers employ mechanical impact heads, inkjet streams, or photocopying techniques, as in laser printers, to perform the printing. It produces printers capable of printing as **many as 10,000** lines per minute.
- This is a tremendous speed for a mechanical device but is still very slow compared to the electronic speed of a processor unit. Monitors, Speakers, Headphones and projectors are also some of the output devices.
- Some units, such as graphic displays, provide both an output function and an input function. The dual role of input and output of such units are referred with single name as I/O unit in many cases.
- **Speakers, Headphones and projectors are some of the output devices. Storage devices such as hard disk, floppy disk, flash drives** are also used for input as well as output.

Memory Unit

- The function of the memory unit is to store programs and data. There are two classes of storage, called **primary and secondary**. **Primary storage** is a fast memory that operates at electronic speeds.
- Programs must be stored in the memory while they are being executed. The memory contains a large number of semiconductor storage cells, each capable of storing one bit of information.
- These cells are rarely read or written as individual cells but instead are processed in groups of **fixed size called words**.
- The memory is organized so that the contents of one word, containing n bits, can be stored or retrieved in one basic operation.
- To provide easy access to any word in the memory, a distinct address is associated with each word location. Addresses are numbers that identify successive locations.
- A given word is accessed by specifying its address and issuing a control command that starts the storage or retrieval process. The number of bits in each word is often referred to as the word length of the computer.
- Typical **word lengths range from 16 to 64 bits**. The capacity of the memory is one factor that characterizes the size of a computer.

- Programs must reside in the memory during execution. Instructions and data can be written into the memory or read out under the controller of the processor.
- It is essential to be able to access any word location in the memory as quickly as possible. Memory in which any location can be reached in a short and fixed amount of time after specifying its address is **called random-access Memory (RAM)**.
- The time required to access one word is called the **memory access time**. This time is fixed, independent of the location of the word being accessed. It typically ranges from a few nanoseconds (ns) to about 100 ns for modern RAM units.
- The memory of a computer is normally implemented as a Memory hierarchy of three or four levels of semiconductor RAM units with different speeds and sizes.
- The small, fast, RAM units are called **caches**. They are tightly coupled with the processor and are often contained on the same integrated circuit chip to achieve high performance.
- The largest and slowest unit is referred to as the main Memory. Although primary storage is essential, it tends to be expensive.

Thus additional, cheaper, secondary storage is used when large amounts of data and many programs have to be stored, particularly for information that is access infrequently. A wide selection of secondary storage devices is available, including magnetic disks and tapes and optical disks

Arithmetic and Logic Unit(ALU):

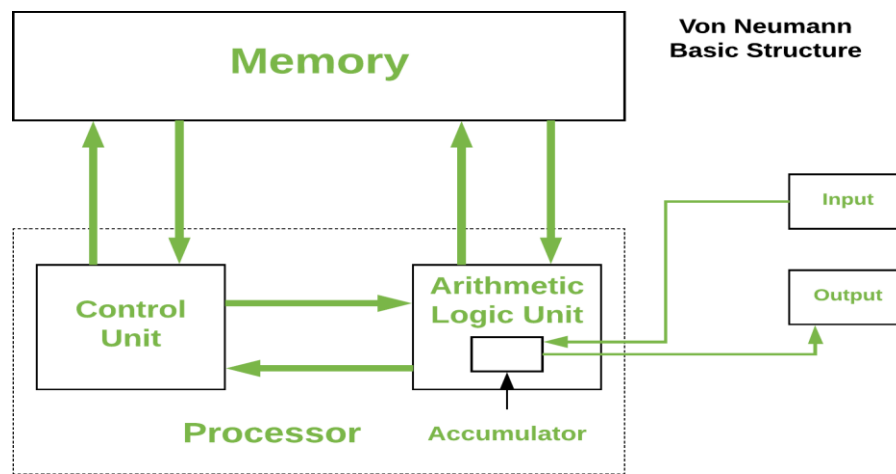
- **ALU** is a digital circuit that performs two types of operations **arithmetic** and **logical**.
- Arithmetic operations are the fundamental mathematical operations consisting of addition, subtraction, multiplication and division. Logical operations consists of comparisons. (i.e) Two pieces of data are compared to see whether one is equal to, less than, or greater than the other.
- The ALU is a fundamental building block of the central processing unit of a computer. Memory enables a computer to store, at least temporarily, data and programs.
- Memory also known as the primary storage or main memory - is a part of the microcomputer that holds data for processing, instructions for processing the data (the program) and information (processed data).
- Part of the contents of the memory is held only temporarily. (i.e)It is stored only as long as the microcomputer is turned on. When you turn the machine off, the contents are lost.
- The control unit instructs the arithmetic-logic unit which operation to perform and then sees that the necessary numbers are supplied. The control and arithmetic & logic units are many times faster than other devices connected to a computer system.

Control Unit (CU):

- It is the part of a CPU that directs its operation. The control unit instructs the rest of the computer system how to carry out a program's instructions.
- It directs the movement of electronic signals between memories, which temporarily holds data, instructions & processed information and the ALU.
- It also directs these control signals between the CPU and input/output devices. The control unit is the circuitry that controls the flow of information through the processor, and coordinates the activities of the other units within it.

VON NEUMANN ARCHITECTURE

- **Von Neumann Architecture** also known as the Von Neumann model, the computer consisted of a CPU, memory and I/O devices.
- The program is stored in the memory. The CPU fetches an instruction from the memory at a time and executes it.
- Thus, the instructions are executed sequentially which is a slow process. Neumann m/c are called control flow computer because instructions are executed sequentially as controlled by a program counter.
- To increase the speed, parallel processing of computer have been developed in which serial CPU's are connected in parallel to solve a problem. Even in parallel computers, the basic building blocks are Neumann processors.
- The von Neumann architecture is a design model for a stored-program digital computer that uses a processing unit and a single separate storage structure to hold both instructions and data.
- It is named after mathematician and early computer scientist John von Neumann.
- Such a computer implements a universal Turing machine, and the common -referential model of specifying sequential architectures, in contrast with parallel architectures.



2. OPERATION AND OPERANDS OF COMPUTER HARDWARE INSTRUCTION

2. Explain about operations operands of computer hardware instruction.

Operation of the computer hardware

- Every computer must be able to perform arithmetic. The MIPS assembly language notation **add a, b, c**
- Instructs a computer to add the two variables b and c and to put their sum in a. This notation is rigid in that each MIPS arithmetic instruction performs only one operation and must always have exactly three variables.
- The following code shows an equivalent MIPS code: **ADD \$s1, \$s2, \$s3** the sum of b and c is placed in a.
- Here, the variables a,b and c are assumed to be stored in the register \$s1, \$s2 and \$s3 all arithmetic immediate value are signed extended.

MIPS Assembly Language

MIPS Operands

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add \$s1,\$s2,\$s3	$\$s1 = \$s2 + \$s3$	Three register operands
	subtract	sub \$s1,\$s2,\$s3	$\$s1 = \$s2 - \$s3$	Three register operands
	add immediate	addi \$s1,\$s2,20	$\$s1 = \$s2 + 20$	Used to add constants
Data transfer	load word	lw \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$	Word from memory to register
	store word	sw \$s1,20(\$s2)	$\text{Memory}[\$s2 + 20] = \$s1$	Word from register to memory
	load half	lh \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$	Halfword memory to register
	load half unsigned	lhu \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$	Halfword memory to register
	store half	sh \$s1,20(\$s2)	$\text{Memory}[\$s2 + 20] = \$s1$	Halfword register to memory
	load byte	lb \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$	Byte from memory to register
	load byte unsigned	lbu \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$	Byte from memory to register
	store byte	sb \$s1,20(\$s2)	$\text{Memory}[\$s2 + 20] = \$s1$	Byte from register to memory
	load linked word	ll \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$	Load word as 1st half of atomic swap
	store condition. word	sc \$s1,20(\$s2)	$\text{Memory}[\$s2+20]=\$s1; \$s1=0 \text{ or } 1$	Store word as 2nd half of atomic swap
load upper immed.	lui \$s1,20	$\$s1 = 20 * 2^{16}$	Loads constant in upper 16 bits	
Logical	and	and \$s1,\$s2,\$s3	$\$s1 = \$s2 \& \$s3$	Three reg. operands; bit-by-bit AND
	or	or \$s1,\$s2,\$s3	$\$s1 = \$s2 \$s3$	Three reg. operands; bit-by-bit OR
	nor	nor \$s1,\$s2,\$s3	$\$s1 = \sim(\$s2 \$s3)$	Three reg. operands; bit-by-bit NOR
	and immediate	andi \$s1,\$s2,20	$\$s1 = \$s2 \& 20$	Bit-by-bit AND reg with constant
	or immediate	ori \$s1,\$s2,20	$\$s1 = \$s2 20$	Bit-by-bit OR reg with constant
	shift left logical	sll \$s1,\$s2,10	$\$s1 = \$s2 \ll 10$	Shift left by constant
shift right logical	srl \$s1,\$s2,10	$\$s1 = \$s2 \gg 10$	Shift right by constant	
Conditional branch	branch on equal	beq \$s1,\$s2,25	if ($\$s1 == \$s2$) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne \$s1,\$s2,25	if ($\$s1 \neq \$s2$) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt \$s1,\$s2,\$s3	if ($\$s2 < \$s3$) $\$s1 = 1$; else $\$s1 = 0$	Compare less than; for beq, bne
	set on less than unsigned	sltu \$s1,\$s2,\$s3	if ($\$s2 < \$s3$) $\$s1 = 1$; else $\$s1 = 0$	Compare less than unsigned
	set less than immediate	slti \$s1,\$s2,20	if ($\$s2 < 20$) $\$s1 = 1$; else $\$s1 = 0$	Compare less than constant
	set less than immediate unsigned	sltiu \$s1,\$s2,20	if ($\$s2 < 20$) $\$s1 = 1$; else $\$s1 = 0$	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr \$ra	go to \$ra	For switch, procedure return
	jump and link	jal 2500	$\$ra = PC + 4$; go to 10000	For procedure call

Name	Example	Comments
32 registers	\$s0-\$s7, \$t0-\$t9, \$zero, \$a0-\$a3, \$v0-\$v1, \$gp, \$fp, \$sp, \$ra, \$at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic, register \$zero always equals 0, and register \$at is reserved by the assembler to handle large constants.
2 ³⁰ memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers.

- All arithmetic operations have exactly three operands, no more and no less, conforms have this conforms to the philosophy of keeping the hardware simple. This situation illustrates the first of four underlying principles of hardware design.

Design Principle 1: Simplicity favors regularity.

Compiling Two C Assignment Statements into MIPS

Example 1:

- This segment of a C program contains the five variables a, b, c, d, and e. Since Java evolved from C, this example and the next few work for either high-level programming language:

```
a = b + c;
d = a - e;
```

Answer

- The translation from C to MIPS assembly language instructions are performed by the compiler. Show the MIPS code produced by a compiler.
- A MIPS instruction operates on two source operands and places the result in one destination operand.
- Hence, the two simple statements above compile directly into these two MIPS assembly language instructions:

```
add a, b, c
sub d, a, e
```

Compiling a complex C Assignment into MIPS

Example 2:

- A somewhat complex statement contains the five variables f, g, h, i, and j:
- What might a C compiler produce?

```
f = (g + h) - (i + j);
```

Answer

- The compiler must break this statement into several assembly instructions, since only one operation is performed per MIPS instruction.
- The first MIPS instruction calculates the sum of g and h. We must place the result somewhere, so the compiler creates a temporary variable, called t0:

```
add t0,g,h # temporary variable t0 contains g + h
```

- Although the next operation is subtract, we need to calculate the sum of i and j before we can subtract.
- Thus, the second instruction places the sum of i and j in another temporary variable created by the compiler, called t1:

```
add t1,i,j # temporary variable t1 contains i + j
```

- Finally, the subtract instruction subtracts the second sum from the first and places the difference in the variable f, completing the compiled code:

```
sub f,t0,t1 # f gets t0 - t1, which is (g + h) - (i + j)
```

Note : ‘ #’symbol indicate the comment line

Operands of the Computer Hardware

Over View

- **Memory operand**
- **Constant or immediate operands**
- **Index register**
- In MIPS instruction set architecture, operand can either in register or memory. Most of the arithmetic and logical instructions use register operands.
- Registers are limited number of special location built directly in hardware and they are visible to the programmer when the computer is completed.
- The size of a register in the MIPS architecture is 32 bits; groups of 32 bits occur so frequently that they are given the name **word** in the MIPS architecture.
- **Word**the natural unit of access in a computer, usually a group of 32 bits; corresponds to the size of a register in the MIPS architecture.
- The reason for the limit of 32 registers may be found in the second of our four underlying design principles of hardware technology:

Design Principle 2: Smaller is faster.

- A very large number of registers may increase the clock cycle time simply because it takes electronic signals longer when they must travel farther.
- Use fewer register to conserve energy.

Example:

Compiling a C Assignment Using Registers

- It is the compiler’s job to associate program variables with registers. Take, for instance, the assignment statement from our earlier example:

```
f = (g + h) - (i + j);
```

- The variables f, g, h, i, and j are assigned to the registers \$s0, \$s1, \$s2, \$s3, and \$s4, respectively. What is the compiled MIPS code?

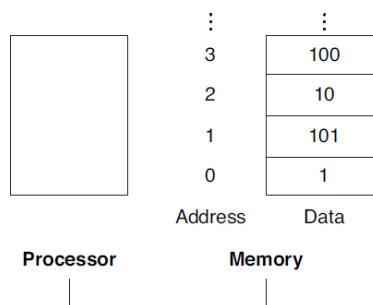
Answer

- The compiled program is very similar to the prior example, except we replace the variables with the register names mentioned above plus two temporary registers, \$t0 and \$t1, which correspond to the temporary variables above:

```
add $t0,$s1,$s2 # register $t0 contains g + h
add $t1,$s3,$s4 # register $t1 contains i + j
sub $s0,$t0,$t1 # f gets $t0 - $t1, which is (g + h)-(i + j)
```

Memory Operands

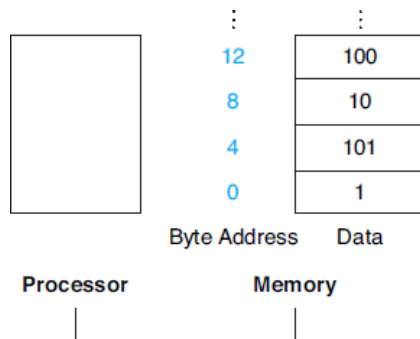
- Programming languages have simple variables that contain single data elements, as in these examples, but they also have more complex data structures—arrays and structures.
- These complex data structures can contain many more data elements than there are registers in a computer.
- The processor can keep only a small amount of data in registers, but computer memory contains billions of data elements.
- Hence, data structures (arrays and structures) are kept in memory.
- As explained above, arithmetic operations occur only on registers in MIPS instructions; thus, MIPS must include instructions that transfer data between memory and registers. Such instructions are called **data transfer instructions**.
- To access a word in memory, the instruction must supply the memory **address**.
- Memory is just a large, single-dimensional array, with the address acting as the index to that array, starting at 0. For example, in the following Figure, the address of the third data element is 2, and the value of Memory[2] is 10



Memory addresses and contents of memory at those locations

- The data transfer instruction that copies data from memory to a register is traditionally called **load**.
- The format of the load instruction is the name of the operation followed by the register to be loaded, then a constant and register used to access memory.

- The sum of the constant portion of the instruction and the contents of the second register forms the memory address. The actual MIPS name for this instruction is **lw**, standing for **load word**.



Actual MIPS memory addresses and contents of memory for those words.

Alignment restriction

- In MIPS, words must start at addresses that are multiples of 4. This requirement is called an **alignment restriction**
- alignment restriction A requirement that data be aligned in memory on natural boundaries .many architecture have alignment restriction

Big endian and little Endian

- 8 bit bytes are divided into two parts:
- Address of the left most byte is called -big endian|| and right most byte is called -little endian -

Compiling an Assignment When an Operand Is in Memory

Example 1:

- Let's assume that A is an array of 100 words and that the compiler has associated the variables g and h with the registers \$s1 and \$s2 as before.
- Let's also assume that the starting address, or base address, of the array is in \$s3. Compile this C assignment statement:

```
g = h + A[8];
```

MIPS Code

- In the given statement, there is a single operation. Whereas, one of the operands is in memory, so we must carry this operation in two steps:

Step 1: load the temporary register(\$s3) + 8

Step 2: perform addition with h((\$s2)), and store result in g(\$s1)

```
lw    $t0,8($s3) # Temporary reg $t0 gets A[8]
```

```
add   $s1,$s2,$t0 # g = h + A[8]
```

- The constant in a data transfer instruction (8) is called the offset, and the register added to form the address (\$s3) is called the **base register**.

Example 2:

Compiling Using Load and Store

- What is the MIPS assembly code for the C assignment statement below?

$$A[12] = h + A[8];$$

- Assume variable h is associated with register \$s2 and the base address of the array A is in \$s3.

MIPS code

```
lw    $t0,32($s3)  # Temporary reg $t0 gets A[8]
add   $t0,$s2,$t0  # Temporary reg $t0 gets h + A[8]
```

- The final instruction stores the sum into A[12], using 48 (4×12) as the offset and register \$s3 as the base register

```
sw    $t0,48($s3)  # Stores h + A[8] back into A[12]
```

- Load word and store word are the instructions that copy words between memory and registers in the MIPS architecture. Other brands of computers use other instructions along with load and store to transfer data.

Constant or Immediate Operands

- Constant variables are used as one of the operand for many arithmetic operation in MIPS architecture
- The constants would have been placed in memory when the program was loaded.
- To avoid load instruction used in arithmetic instruction we can use one operand is a constant
- This quick add instruction with one constant operand is called add immediate or addi. To add 4 to register \$s3, we just write

Design Principle 3: Make the common case fast.

```
lw    $t0, AddrConstant4($s1)  # $t0 = constant 4
add   $s3,$s3,$t0              # $s3 = $s3 + $t0 ($t0 == 4)
```

- Assuming that \$s1 + AddrConstant4 is the memory address of the constant 4.

```
addi   $s3,$s3,4              # $s3 = $s3 + 4
```

Advantage of constant operands

- It uses less energy
- It performs operation in more fast

Index register

- The register in the data transfer instructions was originally invented to hold an index of an array with the offset used for the starting address of an array. Thus, the base register is also called the index register.

3. INSTRUCTION SET ARCHITECTURE (ISA)

3. Discuss about ISA.

- The addressing methods that are commonly used for accessing operands in memory locations and processor registers are also presented.
- The emphasis here is on basic concepts. We use a generic style to describe machine instructions and operand addressing methods that are typical of those found in commercial processors.
- A sufficient number of instructions and addressing methods are introduced to enable us to present complete, realistic programs for simple tasks.
- These generic programs are specified at the assembly-language level, where machine instructions and operand addressing information are represented by symbolic names.
- A complete instruction set, including operand addressing methods, is often referred to as the instruction set architecture (ISA) of a processor.
- The vast majority of programs are written in high-level languages such as C, C++, or Java.
- To execute a high-level language program on a processor, the program must be translated into the machine language for that processor, which is done by a compiler program.
- Assembly language is a readable symbolic representation of machine language.

Memory Locations and Addresses

- The memory consists of many millions of storage cells, each of which can store a bit of information having the value 0 or 1.
- Because a single bit represents a very small amount of information, bits are seldom handled individually.
- The usual approach is to deal with them in groups of fixed size. For this purpose, the memory is organized so that a group of n bits can be stored or retrieved in a single, basic operation.
- Each group of n bits is referred to as a word of information, and n is called the word length. The memory of a computer can be schematically represented as a collection of words, Modern computers have word lengths that typically range from 16 to 64 bits.
- If the word length of a computer is 32 bits, a single word can store a 32-bit signed number or four ASCII-encoded characters, each occupying 8 bits, as shown in Figure.
- A unit of 8 bits is called a byte. Machine instructions may require one or more words for their representation.

- We will discuss how machine instructions are encoded into memory words in a later section, after we have described instructions at the assembly-language level.
- Accessing the memory to store or retrieve a single item of information, either a word or a byte, requires distinct names or addresses for each location.
- It is customary to use numbers from 0 to $2^k - 1$, for some suitable value of k , as the addresses of successive locations in the memory.
- Thus, the memory can have up to 2^k addressable locations. The 2^k addresses constitute the address space of the computer. For example, a 24-bit address generates an address space of 2^{24} (16,777,216) locations.
- This number is usually written as 16M (16 mega), where 1M is the number 2^{20} (1,048,576). A 32-bit address creates an address space of 2^{32} or 4G (4 giga) locations, where 1G is 2^{30} .

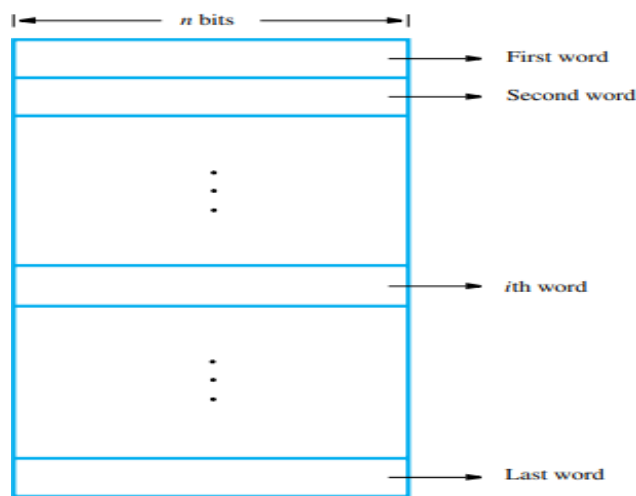


Fig 3.1 Memory words

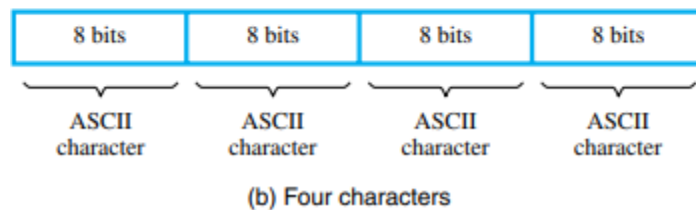
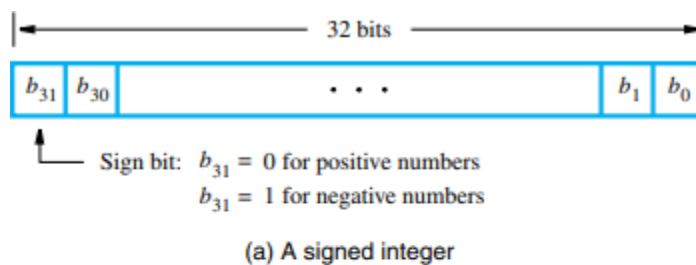


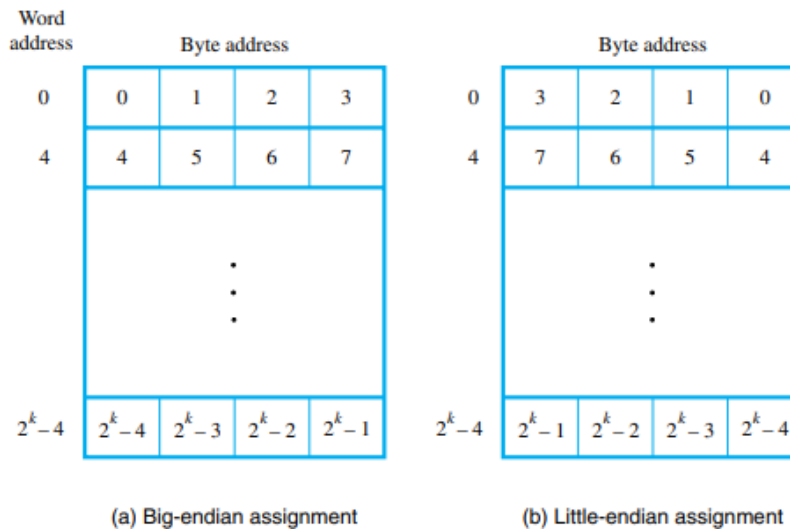
Figure 3.2 Examples of encoded information in a 32-bit word

Byte Addressability

- A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits. It is impractical to assign distinct addresses to individual bit locations in the memory.
- The most practical assignment is to have successive addresses refer to successive byte locations in the memory. This is the assignment used in most modern computers. The term byte-addressable memory is used for this assignment. Byte locations have addresses 0, 1, 2,....
- Thus, if the word length of the machine is 32 bits, successive words are located at addresses 0, 4, 8, , with each word consisting of four bytes.

Big-Endian and Little-Endian Assignments

- The name big-endian is used when lower byte addresses are used for the more significant bytes (the leftmost bytes) of the word.
- The name little-endian is used for the opposite ordering, where the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word.
- The words -more significant|| and -less significant|| are used in relation to the weights (powers of 2) assigned to bits when the word represents a number.
- Both little-endian and big-endian assignments are used in commercial machines. In both cases, byte addresses 0, 4, 8,...., are taken as the addresses of successive words in the memory of a computer with a 32-bit word length.
- These are the addresses used when accessing the memory to store or retrieve a word.



Word Alignment

- In the case of a 32-bit word length, natural word boundaries occur at addresses 0, 4, 8,...., as shown in Figure 2.3. We say that the word locations have aligned addresses if they begin at a byte address that is a multiple of the number of bytes in a word.
- For practical reasons associated with manipulating binary-coded addresses, the number of bytes in a word is a power of 2. Hence, if the word length is 16 (2 bytes), aligned words begin at byte

addresses 0, 2, 4,..., and for a word length of 64 (23 bytes), aligned words begin at byte addresses 0, 8, 16,....

- There is no fundamental reason why words cannot begin at an arbitrary byte address. In that case, words are said to have unaligned addresses.
- But, the most common case is to use aligned addresses, which makes accessing of memory operands more efficient.

Accessing Numbers and Characters

- A number usually occupies one word, and can be accessed in the memory by specifying its word address. Similarly, individual characters can be accessed by their byte address.
- For programming convenience it is useful to have different ways of specifying addresses in program instructions.

Memory Operations

- Both program instructions and data operands are stored in the memory. To execute an instruction, the processor control circuits must cause the word (or words) containing the instruction to be transferred from the memory to the processor.
- Operands and results must also be moved between the memory and the processor. Thus, two basic operations involving the memory are needed, namely, Read and Write.
- The Read operation transfers a copy of the contents of a specific memory location to the processor. The memory contents remain unchanged.
- To start a Read operation, the processor sends the address of the desired location to the memory and requests that its contents be read.
- The memory reads the data stored at that address and sends them to the processor. The Write operation transfers an item of information from the processor to a specific memory location, overwriting the former contents of that location.
- To initiate a Write operation, the processor sends the address of the desired location to the memory, together with the data to be written into that location.
- The memory then uses the address and data to perform the write.

4. INSTRUCTION AND INSTRUCTION SEQUENCING

4. Discuss about instruction and instruction sequencing.

- The tasks carried out by a computer program consist of a sequence of small steps, such as adding two numbers, testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen.

- **A computer must have instructions capable of performing 4 types of operations:**
 - 1) Data transfers between the memory and the registers (MOV, PUSH, POP, XCHG).
 - 2) Arithmetic and logic operations on data (ADD, SUB, MUL, DIV, AND, OR, NOT).
 - 3) Program sequencing and control (CALL, RET, LOOP, INT).
 - 4) I/O transfers (IN, OUT).

REGISTER TRANSFER NOTATION (RTN)

Here we describe the transfer of information from one location in a computer to another. Possible locations that may be involved in such transfers are memory locations, processor registers, or registers in the I/O subsystem.

- Most of the time, we identify such locations symbolically with convenient names.
- The possible locations in which transfer of information occurs are:
 - 1) Memory-location
 - 2) Processor register &
 - 3) Registers in I/O device.

Location	Hardware Binary Address	Example	Description
Memory	LOC, PLACE, NUM	$R1 \leftarrow [LOC]$	Contents of memory-location LOC are transferred into register R1.
Processor	R0, R1, R2	$[R3] \leftarrow [R1] + [R2]$	Add the contents of register R1 & R2 and places their sum into R3.
I/O Registers	DATAIN, DATAOUT	$R1 \leftarrow DATAIN$	Contents of I/O register DATAIN are transferred into register R1.

ASSEMBLY LANGUAGE NOTATION

- To represent machine instructions and programs, assembly language format is used.

Assembly Language Format	Description
Move LOC, R1	Transfer data from memory-location LOC to register R1. The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten.
Add R1, R2, R3	Add the contents of registers R1 and R2, and places their sum into register R3.

BASIC INSTRUCTION TYPES

Instruction Type	Syntax	Example	Description	Instructions for Operation $C \leftarrow [A] + [B]$
Three Address	Opcod Source1, Source2, Destination	Add A, B, C	Add the contents of memory-locations A & B. Then, place the result into location C.	
Two Address	Opcod Source, Destination	Add A, B	Add the contents of memory-locations A & B. Then, place the result into location B, replacing the original contents of this location. Operand B is both a source and a destination.	Move B, C Add A, C
One Address	Opcod Source/Destination	Load A	Copy contents of memory-location A into accumulator.	Load A Add B Store C
		Add B	Add contents of memory-location B to contents of accumulator register & place sum back into accumulator.	
		Store C	Copy the contents of the accumulator into location C.	
Zero Address	Opcod [no Source/Destination]	Push	Locations of all operands are defined implicitly. The operands are stored in a pushdown stack.	Not possible

INSTRUCTION EXECUTION & STRAIGHT LINE SEQUENCING

- The program is executed as follows:

- 1) Initially, the address of the first instruction is loaded into PC.
- 2) Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses. This is called Straight-Line sequencing.
- 3) During the execution of each instruction, PC is incremented by 4 to point to next instruction.

- There are 2 phases for Instruction Execution:

- 1) Fetch Phase: The instruction is fetched from the memory-location and placed in the IR.
- 2) Execute Phase: The contents of IR is examined to determine which operation is to be performed.

The specified-operation is then performed by the processor.

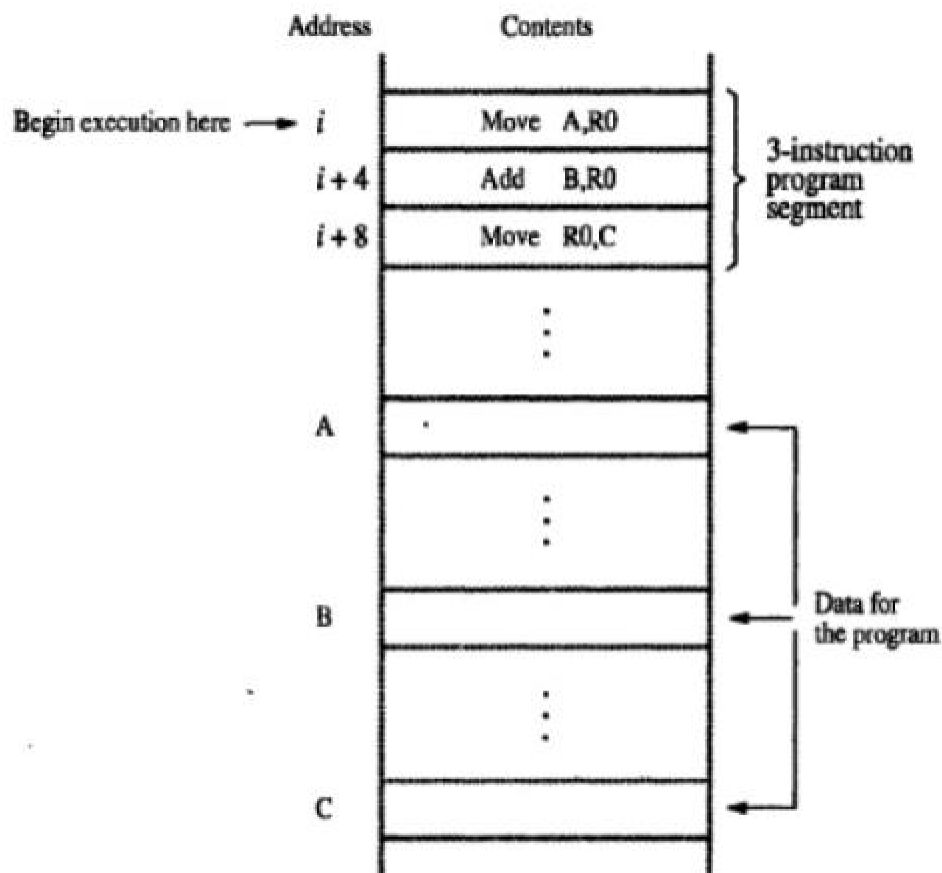


Fig 3.3 A program for $C \leftarrow [A] + [B]$

Program Explanation

- Consider the program for adding a list of n numbers
- The Address of the memory-locations containing the n numbers are symbolically given as NUM1, NUM2.....NUMn.
- Separate Add instruction is used to add each number to the contents of register R0.
- After all the numbers have been added, the result is placed in memory-location SUM.

i	Move NUM1,R0
$i + 4$	Add NUM2,R0
$i + 8$	Add NUM3,R0
	⋮
$i + 4n - 4$	Add NUM n ,R0
$i + 4n$	Move R0,SUM
	⋮
SUM	
NUM1	
NUM2	
	⋮

5. ADDRESSING MODES

5. What is the need for addressing in a computer system? Explain the different addressing modes with suitable examples. (16)Apr 2011, Nov 2011, 2012, 2013,May 2013 (Nov/Dec 2014) (Apr/May 2015) (Nov/Dec 2016) (Or) Explain direct, immediate, relative and indexed addressing modes with examples. Apr/May 2017, 2018, Nov. / Dec. 2018 (Nov/Dec 2019)Nov/Dec 2020.(Or) Identify the addressing mode involved in the instruction XOR r1 r2 + 100 r1 and determine the resultant stored in register R1 if all of its bit were 1'st initially .

Addressing Modes

- The different ways in which the location of an operand is specified in instructions are referred to as addressing modes.
- Different types of addresses involve tradeoffs between instruction length, addressing flexibility and complexity of address calculation.

Overview

The different types of addressing modes are:

- Immediate addressing mode
- Direct or absolute addressing mode
- Indirect addressing mode
- Register addressing mode
- Indexed addressing mode(Displacement)
- Relative addressing mode
- Auto increment
- Auto decrement
- Implied (Stack, and a few others)

Immediate Addressing and Small Operands

- The operand is given explicitly in the instruction.
Example: **MOVE #200, R0**
- The above statement places the value 200 in the register R0. A common convention is to use the sharp sign (#) in front of the value to indicate that this value is to be used as an immediate operand.
- A great many immediate mode instructions use small operands. (8 bits)
- In 32 or 64 bit machines with variable length instructions space is wasted if immediate operands are required to be the same as the register size.
- Some instruction formats include a bit that allows small operands to be used in immediate instructions.

- ALU will zero-extend or sign-extend the operand to the register size.

Instruction



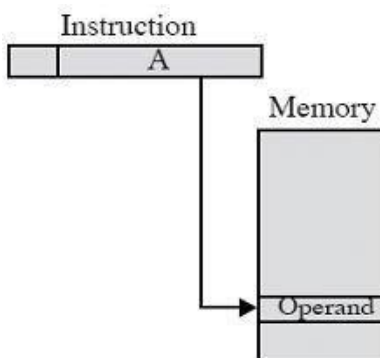
Immediate

Direct Addressing (Absolute addressing mode)

- The operand is in a memory location; the address of this location is given explicitly in the instruction. (In some assembly languages, this mode is called Direct Mode.

Example: **MOVE LOC, R2**

- This instruction copies the contents of memory location of LOC to register R2.
- Address field contains address of operand.
- Effective address (EA) = address field (A).
e.g. add ax, count or add ax,[10FC]
- Look in memory at address for operand.
- Single memory reference to access data.
- No additional calculations to work out effective address.



Direct Addressing

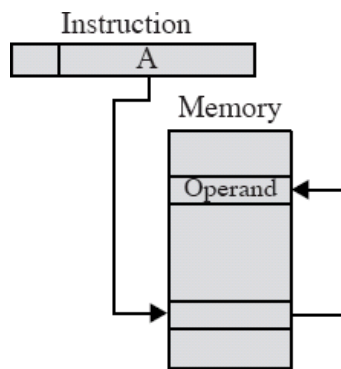
Memory-Indirect Addressing

- The effective address of the operand is the contents of a register or memory location whose address appears in the instruction.

Example **Add (R2),R0**

- Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2.

- The initialization section of the program loads the counter value n from memory location N into $R1$ and uses the immediate addressing mode to place the address value $NUM + 1$, which is the address of the first number in the list, into $R2$.



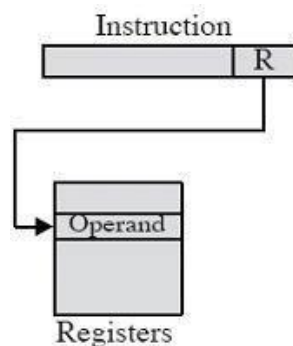
Memory-Indirect Addressing

Register Direct Addressing

- The operand is the contents of a processor register; the name (address) of the register is given in the instruction.

Example: **MOV R1,R2**

- This instruction copies the contents of register $R2$ to $R1$.
- Operand(s) is(are) registers $EA = R$.
- There are a limited number of registers.
- Therefore a very small address field is needed.
- Shorter instructions are used.
- Instruction fetch is faster when compared to other.
- X86: 3 bits used to specify one of 8 registers.

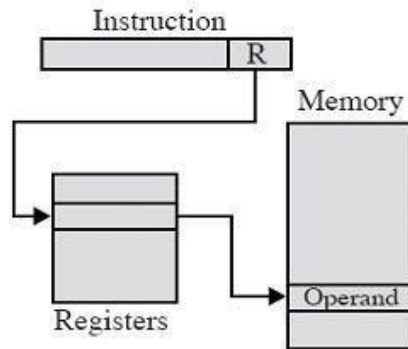


Register Direct Addressing

Register Indirect Addressing

- Similar to memory-indirect addressing; much more common, $EA = (R)$.
- Operand is in memory cell pointed to by contents of register R .
- Large address space ($2n$).

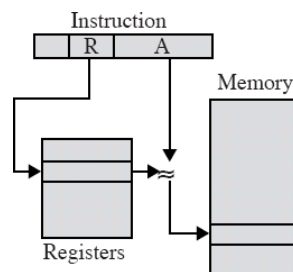
- One fewer memory access than indirect Addressing.



Register Indirect Addressing

Displacement Addressing(Index addressing mode)

- The effective address of the operand is generated by adding a constant value to the contents of a register. The register used may be either a special register provided for this purpose, or, more commonly; it may be anyone of a set of general-purpose registers in the processor.
- In either case, it is referred to as an index register. We indicate the index mode symbolically as **X(Ri)**.
- Where X denotes the constant value contained in the instruction and Ri is the name of the register involved. The effective address of the operand is given by **EA = X + [Ri]**.
- The contents of the index registers are not changed in the process of generating the effective address.
 - $EA = A + (R)$
 - Combines register indirect addressing with direct addressing
 - Address field hold two values
 - A = base value
 - R = register that holds displacement
 - or vice versa



Displacement Addressing

Types of Displacement Addressing

- Relative Addressing

- Base-register addressing
- Indexing

Relative Addressing

- We have defined the Index mode using general-purpose processor registers. A useful version of this mode is obtained if the program counter, PC, is used instead of a general purpose register.
- Then, X(PC) can be used to address a memory location that is X bytes away from the location presently pointed to by the program counter.
- Since the addressed location is identified "**relative**" to the program counter, which always identifies the current execution point in a program, the name Relative mode is associated with this type of addressing.
- $EA = X + (PC)$

Base-Register Addressing

- A holds displacement.
- R holds pointer to base address.
- R may be explicit or implicit.
- e.g. segment registers in 80x86 are base registers and are involved in all EA computations.
- x86 processors have a wide variety of base addressing formats.

Auto- increment mode

- The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this registers are automatically incremented to point to the next item in a list.
- We denote the Auto increment mode by putting the specified register in parentheses, to show that the contents of the registers are used as the effective address, followed by a plus sign to indicate that these contents are to be incremented after the operand is accessed.
- Thus, the Auto increment mode is written as,

$$(Ri) +$$

- As a companion for the Autoincrement mode, another useful mode accesses the items of a list in the reverse order:

Auto- decrement mode

- The contents of a register specified in the instructions are first automatically decremented and is then used as the effective address of the operand.
- We denote the Autodecrement mode by putting the specified register in parentheses, preceded by a minus sign to indicate that the contents of the registers are to be decremented before being used as the effective address. Thus, we write

Stack Addressing

- Operand is (implicitly) on top of stack.
e.g. PUSH, POP

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	Ri	EA = Ri
Absolute (Direct)	LOC	EA = LOC
Indirect	(Ri)	EA = [Ri]
	(LOC)	EA = [LOC]
Index	X(Ri)	EA = [Ri] + X
Base with index	(Ri,Rj)	EA = [Ri] + [Rj]
Base with index and offset	X(Ri,Rj)	EA = [Ri] + [Rj] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(Ri)+	EA = [Ri]; Increment Ri
Autodecrement	-(Ri)	Decrement Ri; EA = [Ri]

EA = effective address

Value = a signed number

- X87 is a stack machine so it has instructions such as,
FADDP; st(1) <- st(1) + st(0); pop stack; result left in st(0)
FIMUL qword ptr [bx]; st(0) <- st(0) * 64 integer pointed to; by bx

6. Consider the computer with three instruction classes and CPI measurements as given below and instruction counts for each instruction class for the same program from two different compilers are given. Assume that the computer's clock rate is 4 GHZ. Which code sequence will execute faster according to execution time? (6) (NOV/DEC2014)

Code from	CPI for the instruction class		
	A	B	C
CPI	1	2	3
Code from	Instruction count for each class		
	A	B	C
Compiler 1	2	1	2
Compiler 2	4	1	1

ANSWER

- Sequence 1 executes $2 + 1 + 2 = 5$ instructions. Sequence 2 executes $4 + 1 + 1 = 6$ instructions. Therefore, sequence 1 executes fewer instructions. We can use the equation for CPU clock cycles based on instruction count and CPI to find the total number of clock cycles for each sequence:

$$\text{CPU clock cycles} = \sum_{i=1}^n (\text{CPI}_i \times C_i)$$

- This yields,
- CPU clock cycles₁ = (2 × 1) + (1 × 2) + (2 × 3) = 2 + 2 + 6 = 10 cycles
- CPU clock cycles₂ = (4 × 1) + (1 × 2) + (1 × 3) = 4 + 2 + 3 = 9 cycles
- So code sequence 2 is faster, even though it executes one extra instruction. Since code sequence 2 takes fewer overall clock cycles but has more instructions, it must have a lower CPI. The CPI values can be computed by,

$$\text{CPI} = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$$

$$\text{CPI}_1 = \frac{\text{CPU clock cycles}_1}{\text{Instruction count}_1} = \frac{10}{5} = 2.0$$

$$\text{CPI}_2 = \frac{\text{CPU clock cycles}_2}{\text{Instruction count}_2} = \frac{9}{6} = 1.5$$

6. ENCODING OF MACHINE INSTRUCTION

7. Explain about encoding of machine instruction.

- We have introduced a variety of useful instructions and addressing modes. These instructions specify the actions that must be performed by the processor circuitry to carry out the desired tasks.
- We have often referred to them as machine instructions. Actually, the form in which we have presented the instructions is indicative of the form used in assembly languages, except that we tried to avoid using acronyms for the various operations, which are awkward to memorize and are likely to be specific to a particular commercial processor.
- To be executed in a processor, an instruction must be encoded in a compact binary pattern. Such encoded instructions are properly referred to as machine instructions.
- The instructions that use symbolic names and acronyms are called assembly language instructions, which are converted into the machine instructions using the assembler program.
- We have seen instructions that perform operations such as add, subtract, move, shift, rotate, and branch. These instructions may use operands of different sizes, such as 32-bit and 8-bit numbers or 8-bit ASCII-encoded characters.
- The type of operation that is to be performed and the type of operands used may be specified using an encoded binary pattern referred to as the OP code for the given instruction.
- Suppose that 8 bits are allocated for this purpose, giving 256 possibilities for specifying different instructions. This leaves 24 bits to specify the rest of the required information.
- Let us examine some typical cases. The instruction

Add R1, R2

- Has to specify the registers R1 and R2, in addition to the OP code.
- If the processor has 16 registers, then four bits are needed to identify each register. Additional bits are needed to indicate that the Register addressing mode is used for each operand.

- The instruction

Move 24(R0), R5

- Requires 16 bits to denote the OP code and the two registers, and some bits to express that the source operand uses the Index addressing mode and that the index value is 24.

- The shift instruction

LShiftR #2, R0

- And the move instruction

Move #\$3A, R1

- Have to indicate the immediate values 2 and #\$3A, respectively, in addition to the 18 bits used to specify the OP code, the addressing modes, and the register.

- This limits the size of the immediate operand to what is expressible in 14 bits

- Consider next the branch instruction

Branch >0 LOOP

- Again, 8 bits are used for the OP code, leaving 24 bits to specify the branch offset.
- Since the offset is a 2's-complement number, the branch target address must be within 223 bytes of the location of the branch instruction.
- To branch to an instruction outside this range, a different addressing mode has to be used, such as Absolute or Register Indirect. Branch instructions that use these modes are usually called Jump instructions.

- In all these examples, the instructions can be encoded in a 32-bit word. Depicts a possible format.

- There is an 8-bit Op-code field and two 7-bit fields for specifying the source and destination operands. The 7-bit field identifies the addressing mode and the register involved (if any).

- The -Other info|| field allows us to specify the additional information that may be needed, such as an index value or an immediate operand.

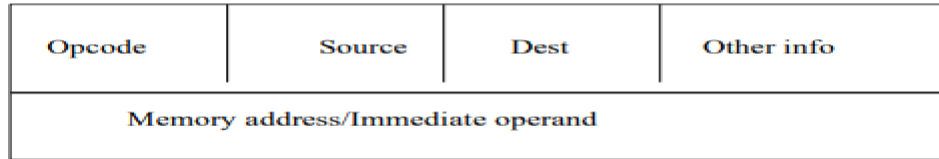
- But, what happens if we want to specify a memory operand using the Absolute addressing mode?

- The instruction Move R2, LOC

(a) One-word instruction



(b) Two-Word instruction



(c) Three-operand instruction

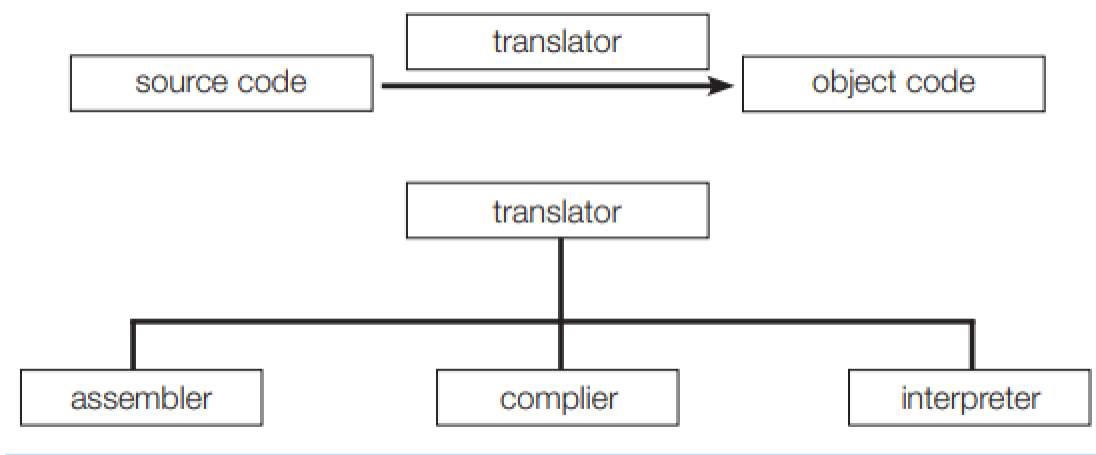


- Requires 18 bits to denote the OP code, the addressing modes, and the register.
- This leaves 14 bits to express the address that corresponds to LOC, which is clearly insufficient. And #FF000000. R2
- In which case the second word gives a full 32-bit immediate operand. If we want to allow an instruction in which two operands can be specified using the Absolute addressing mode, for example
Move LOC1, LOC2
- Then it becomes necessary to use an additional words for the 32-bit addresses of the operands. This approach results in instructions of variable length, dependent on the number of operands and the type of addressing modes used.
- Using multiple words, we can implement quite complex instructions, closely resembling operations in high-level programming languages.
- The term complex instruction set computer (CISC) has been used to refer to processors that use instruction sets of this type.
- The restriction that an instruction must occupy only one word has led to a style of computers that have become known as reduced instruction set computer (RISC).
- The RISC approach introduced other restrictions, such as that all manipulation of data must be done on operands that are already in processor registers.
- This restriction means that the above addition would need a two-instruction sequence
Move (R3), R1 Add R1, R2
- If the Add instruction only has to specify the two registers, it will need just a portion of a 32-bit word. So, we may provide a more powerful instruction that uses three operands
Add R1, R2, R3
- Which performs the operation $R3 \rightarrow [R1] + [R2]$
- A possible format for such an instruction is shown in fig c. Of course, the processor has to be able to deal with such three-operand instructions.
- In an instruction set where all arithmetic and logical operations use only register operands, the only memory references are made to load/store the operands into/from the processor registers.
- RISC-type instruction sets typically have fewer and less complex instructions than CISC-type sets.

7. INTERACTION BETWEEN ASSEMBLY AND HIGH LEVEL LANGUAGE

8. Explain the concept of interaction between assembly and high level language

- What are 'high level languages'? High level languages are called 'high-level' because they are closer to human languages and are further removed from machine languages than assembly language.
- There is no one-to-one relationship between the instructions in a high level language and machine language as there is with assembly language.
- List three examples of a high level language. Basic, C, Fortran, Python, Ada etc.
- List three advantages of assembly language over a high level language.
- It requires less memory and execution time.
- It allows hardware-specific complex jobs in an easier way.
- It is suitable for time-critical jobs.
- It is most suitable for writing interrupt service routines and other memory resident programs.
- List three advantages of using a high level language over assembly language.
- Faster program development – it is less time consuming to write and then test the program.
- It is not necessary to remember the registers of the CPU and mnemonic instructions.
- Portability of a program from one machine to other.
- Each assembly language is specific to a particular type of CPU, but most high-level programming languages are generally portable across multiple architectures.
- A compiler reads the whole high level code and translates it into a complete machine code program which is output as a new file and can be saved.
- The biggest advantage of this is that the translation is done once only and as a separate process. The program that is run is already translated into machine code so is much faster in execution.
- The disadvantage is that you cannot change the program without going back to the original source code, editing that and recompiling.
- An interpreter reads the source code one instruction or line at a time, converts this line into machine code and executes it.
- The machine code is then discarded and the next line is read. The advantage of this is it's simple and you can interrupt it while it is running, change the program and either continue or start again.
- The disadvantage is that every line has to be translated every time it is executed, even if it is executed many times as the program runs. And because of this interpreters tend to be slow.



1. What are the five components of computer system? Apr/May 2017, 2019

The five classic components of computers are input unit, output unit, memory unit, arithmetic & logic unit and control unit.

2. What is cache memory?

The small and fast RAM units are called as caches.

When the execution of an instruction calls for data located in the main memory, the data are fetched and a copy is placed in the cache.

- Later if the same data are required it reads directly from the cache.

3. What is the function of ALU?

- Most of the computer operations (arithmetic & logic) are performed in ALU. The data required for the operation is brought by the processor and the operation is performed by the ALU.

4. What is the function of control unit?

- The Control unit is the main part of the computer that coordinates the entire computer operations. Data transfers between the processor and memory controlled by the control unit through timing signal.

5. What are basic operations of a computer memory?

- The basic operations of the memory are READ and WRITE.
 - READ – read the data from input device to memory.
 - WRITE – writes data to the output device.

6. List out the operations of the computer.

The computer accepts the information in the form of programs and data through an input unit and stores it in the memory.

1. Information stored in the memory is fetched under program control into an arithmetic and logic unit where it is processed.
2. Processed information leaves the computer through an output unit.
3. All activities inside the machines are directed by the control unit.

7. What are the main elements of a computer?

- **Processor:** To interpret and execute programs.
- **Memory:** For storing programs and data.
- **Input-output equipment:** For transferring information between the computer and outside world.

8. Define Computer design.

- It is concerned with the hardware design of the computer. Once the computer specifications are formulated, it is the task of the designer to develop hardware for the system.

- Computer design is concerned with the determination of what hardware should be used and how the parts should be connected. This aspect of computer hardware is sometimes referred to as computer implementation.

9. What is instruction set architecture?

- An abstract interface between the hardware and the lowest level software that encompasses all the information necessary to write a machine language program that will run correctly.
- Including instructions, registers, memory access, I/O and so on.

10. State Amdahl's law. Nov / Dec 2014

- Amdahl's Law is used to find the execution time of a program after making the improvement. It can be represented in an equation as follows:

$$\text{Execution time after improvement} = \frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

- Hence, Amdahl's Law can be used to estimate performance improvements.

11. Define Stored Programmed Concept.

- Storing program and their data in the same high-speed memory.
- It enables a program to modify its own instructions (such self-modifying Programs have undesirable aspects, however and are rarely used).

12. What are the registers generally contained in the processor?(Nov/Dec-2019)

- MAR – Memory Address Register.
- MDR – Memory Data Register.
- IR – Instruction Register.
- R₀ – R_n – General purpose Register.
- PC – Program Counter.

13. What do you mean by Memory address register (MAR) and Memory dataregister (MDR)?

- The MAR holds the address of the location to be accessed.
- The MDR contains the data to be written into or read out of the addressed location.

14. What is Data path?

- The component of the processor that performs arithmetic operations is called data path.

15. What is elapsed time of computer system?

- The total time to execute the total program is called elapsed time.
- It is affected by the speed of the processor, the disk and the printer.

16. What is processor time of a program?

- The period during which the processor is active is called processor time of a program.

- It depends on the hardware involved in the execution of individual machine instructions.

17. Define clock rate.

- The clock rate is given by,

$$R=1/P,$$

- Where P is the length of one clock. It can be measure as cycles per second (Hertz).

18. What is meant by clock cycle?

- Processor circuit is controlled by a timing signal called a clock.
- The clock defines regular time intervals, called clock cycle.
- To execute the machine instruction the processor divides the action to be performed into sequence of basic steps. Each step can be completed in one clock cycle.

19. Write down the basic performance equation. (Apr/May-2014)(Nov/Dec 2019)

$$T=N*S/R$$

Where

T-Processor time

N-Number of machine instructions

S-Number of basic steps needed to execute one machine instruction

R-Clock rate

20. What is meant by addressing mode? List its types. (May/June 2013) Nov/ Dec 2013

The addressing mode is defined as the different ways in which the location or of an operand is specified in an instruction.

The different types of addressing modes are:

1. Immediate addressing mode
2. Register addressing mode
3. Direct or absolute addressing mode
4. Indirect addressing mode
5. Indexed addressing mode
6. Relative addressing mode
7. Auto increment
8. Auto decrement

21. Define Register addressing mode with an example.

- In register addressing mode, the operand is the content of a processor register. The name (address) of the register is given in the instruction.

Effective address (EA) = Ri, Where Ri is a processor register.

22. Define absolute addressing mode with an example.

- In absolute addressing mode, the operand is in a memory location. The addresses of this location are given explicitly in the instruction. This is also called as direct addressing mode.

EA = Loc Where loc is the memory address.

23. What is relative addressing mode with an example? (N/D 2014)

- The Effective address is determined by the index mode using the program counter in place of general purpose register. This mode is used to access the data operands.

EA = X + [PC]

24. What is indirect addressing mode?

- The Effective address of the operand is the contents of a register or memory location whose address appears in the instruction.

EA = [Ri] or EA = [Loc]

25. What is indexed addressing mode?

- The Effective address of the operand is generated by adding a constant value to the contents of a register.

EA = X + [Ri].

26. Define auto increment mode of addressing.

The Effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this registers are automatically incremented to point to the next item in the list.

EA = (Ri) +

27. Define auto decrement mode of addressing.

The contents of a register specified in the instructions are first automatically decremented and are then used as the effective address of the operand.

EA = - (Ri)

28. List the basic instruction types. May / June 2013

The various instruction types are,

- Three address instructions
- Two-address instructions
- Single-address instructions
- Zero-address instructions

29. What is register?

- A small set of high-speed storage devices called registers, which serve as implicit storage locations for operands and results.

30. List the phases, which are included in the each instruction cycle?

- **Fetch:** Fetches instruction from main memory (M).
- **Decode:** Decodes the instruction's opcode.
- **Load:** Loads (read) from M any operands needed unless they are already in CPU Registers.
- **Execute:** Executes the instruction via a register-to-register operation using an appropriate functional unit of the CPU such as a fixed-point adder.
- **Store:** Stores (write) the results in M unless they are to be retained in CPU register.

31. What are the types of computer?

- Mini computer
- Micro computers
- Mainframe computers
- Super computers

32. What are the two major steps in processing an instruction? (Or) Write the two steps that are common to implement any type of instruction. Nov. / Dec. 2018

- **Fetch step:** During this step a new instruction is read from the external memory M by the CPU.
- **Execute step:** During this step operations specified by the instructions are executed by the CPU.

33. What are the speedup techniques available to increase the performance of a computer?

- **Cache:** It is a fast accessible memory often placed on the same chip as the CPU. It is used to reduce the average time required to access an instruction or data to a single clock cycle.
- **Pipelining:** Allows the processing of several instructions to be partially overlapped.
- **Super scalar:** Allows processing of several instructions in parallel (full overlapping).

34. What are Timing signals?

- Timing signals are signals that determine when a given action is to take place.
- Data transfers between the processor and the memory are also controlled by the control unit through timing signals.

35. Distinguish between auto increment and auto decrement addressing mode. (May/June 2016)

Auto increment	Auto decrement
1.The effective address of the operand is the contents of the register specified in the instruction. After accessing the operand, the contents of the register are incremented to address the next location.	1.The contents of a register specified in the instruction are decremented and then are used as effective address to access a memory location.
2.Auto increment is symbolically represented as $(R_i)+$. Example: move(R_2), R_0+	2.Auto decrement mode is symbolically represented as $-(R_i)$. Example: Move R_1 , $-(R_0)$

36. What is an opcode? How many bits are needed to specify 32 distinct operations? (Apr/May 2011)

- An **opcode** is the first byte of an instruction in machine language which tells the hardware what operation needs to be performed with this instruction.
- Every processor/controller has its own set of opcodes defined in its architecture. Opcode is the operation to be performed on data. An opcode is followed by data like address, values etc if needed.
- 5 bits are needed to specify 32 distinct operations.

37. Define word length. (Nov/Dec 2011)

- In computer architecture, a **word** is a unit of data of a defined bit length that can be addressed and moved between storage and the computer [processor](#).
- Address that is divided by 4 is called word.
- The number of bits in the word is called **word length**.
- In longer architected **word length**,the computer processor can do more in a single operation.

38. What are the merits and demerits of single address instructions? (Nov/Dec 2011)

Single address instruction,

Eg: **Add A**

Store A

Add the contents of memory location A to the contents of the accumulator register and place the sum back into accumulator.

39. Explain the disadvantages of using a single type of instruction.

In practice the codes in an instruction (opcode and condition) may be fairly small e.g. 2. to .8 bits. However, if the instruction is to be able to reference large quantities of data then the addresses must be large e.g. 16..32 bits. If the above instruction were to use 6 bits for the opcode, 4 bits for the condition code and 16 bits for each address then it would have to be 90 bits long.

40. What is relative addressing mode? When is it used? (May/June 2012)

- The effective address is determined by the index mode using program counter in place of the general purpose registers.
- This address is commonly used to specify the target address in branch instruction.
 - **Example: JNZ BACK**
 - This instruction causes program executive to go to the branch target location identified by the name BACK, if the branch condition is satisfied.

41. Suppose you wish to run a program P with 8.5×10^9 instructions on a 5 Ghz machine with CPI of 0.8. What is the expected CPU time? (Nov/Dec 2010)

Percentage of elapsed time = (User CPU Time + System CPU Time) / Elapsed Time

Expected CPU time = $0.8 - 0.2 \times 8.5 \times 10^9 = 1.36$

42. What does the term hertz refer to? (Nov/Dec 2010)

- The hertz abbreviated as Hz.
- It is a unit of frequency.
- It is equal to 1 cycle per second.

43. Mention the registers used for communications between processor and main memory. (May/June 2010)

- 1) **MAR(Memory Address Register):** The **Memory Address Register (MAR)** is a CPU register that either stores the memory address from which data will be fetched to the CPU or the address to which data will be sent and stored.
- 2) **MDR (Memory Data Register):** It is the register of a computer's control unit that contains the data to be stored in the computer storage (e.g. RAM), or the data after a fetch from the computer storage. It acts like a buffer and holds anything that is copied from the memory ready for the processor to use it.

44. What is SPEC? Specify the formula for SPEC rating. (May/June 2012)(Apr/May 2014)

- SPEC is a nonprofit consortium of 22 major computer vendors whose common goals are -to provide the industry with a realistic yardstick to measure the performance of advanced computer systems and to educate consumers about the performance of vendors' products.
- SPEC creates, maintains, distributes, and endorses a standardized set of application-oriented programs to be used as benchmarks.

The formula for SPEC rating is as follows:

SPEC rating (ratio) = TR / TC;

where,

TR = Running time of the Reference Computer;

TC = Running time of the Computer under test;

If the SPEC rating = 50 means that the computer under test is 50 times as fast as the ultra sparc 10. This is repeated for all the programs in the SPEC suit, and the geometric mean of the result is computed.

45. Give an example each of zero-address, one-address, two-address and three-address instructions. (Or) Classify the instructions based on the operations they perform and give one example to each category. Apr. / May 2018, Nov. / Dec. 2018

- Zero address- push (Push the value as top of stock)
- One address- INC CL (If carry set, increment CL by one)
- Two address- Add A,B ($A \rightarrow A+B$)
- Three address- Add A,B,C ($A \rightarrow B+C$)

46. Which data structures can be best supported using (a) indirect addressing mode (b) indexed addressing mode?

- (a) Indirect addressing mode – Pointer data structure
- (b) Indexed addressing mode- Array data structure

47. What are the four basic types of operations that need to be supported by an instructor set?

- (i) Data transfer between memory and the processor register.
- (ii) Arithmetic and logic operations on Data.
- (iii) Program sequencing and control.
- (iv) I/O transfer.

48. What are the address-sequencing capabilities required in a control memory?

- (i) Incrementing of the control address register.
- (ii) Unconditional branch as specified by address field of the micro instruction.
- (iii) Conditional branch depending on status bits in registers of computer.
- (iv) A facility for sub-routines calls and returns.

49. What are the limitations of assembly language? (M/J 2007)

- (i) Assembly language is converted to Machine language using assembler which is time consuming when compared with machine language.
- (ii) It is difficult to solve the complex problems.
- (iii) A set of symbolic names (mnemonics) and rules has to be followed.

50. A memory byte location contains the pattern 00101100. What does this pattern represent when interpreted as a number? What does it represent as an ASCII Code? (Nov/Dec 2007)

- Interpreted number is 44.
- ASCII code is NULL/idle.

51. What is the information conveyed by addressing modes? (Nov/Dec 2007)

- The information conveyed by addressing mode is to specify the location of an operand in an instruction.

52. Why is the data bus in most microprocessors bi-directional while the address bus is unidirectional? (Apr/May 2008)

- The data bus is bi-directional bus and is used to fetch instruction from memory and to send a command to an I/O device or port. Address is unidirectional to carry memory address while reading from or writing into memory.

53. What is meant by the stored program concept? Discuss. (May/June 2007)

- A set of instruction that performs a task is called a program. Usually the program is stored in the memory. The processor fetches the instructions that take up the program from the memory, one at a time and perform the desired operation.

54. What are the two techniques used to increase the clock rate R?

The two techniques used to increase the clock rate R are:

- The Integrated Circuit (IC) technology can be increased which reduces the time needed to complete a basic step.
- We can reduce the amount of processing done in one basic step.

55. What is Big-Endian and Little-Endian representations? (Nov/Dec 2014)

- **The big-endian** is used when lower byte addresses are used for the more significant bytes(The leftmost bytes) of the word.
- **The little-endian** is used for the opposite ordering, where the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word.

56. What is meant by instructions? (May/June 2016)

- Instructions are command that is governed by the transfer of information within the computer as well as between the computers and its input and output device.

57. What is the use of Instruction register?

- It holds the instructions that are currently being executed.

58. What is the use of MAR?

- It holds the address of the location to be accessed.

59. What is the use of MDR?

- It holds the data to be written into or read out of the addressed location.

60. State the basic performance equation of a computer. (Apr/May 2014)

$$T = (N \times S) / R$$

Where,

N- Number of instructions

S- Average numbers of steps needed to execute one instruction.

R- Clock rate.

61. What are the two basic operations involving in the memory?

1. Load (Read or fetch)
2. Store (Write)

62. How to measure the performance of the system?

1. Response time
2. Throughput.

63. What is register indirect addressing mode? When is it used? (Nov/Dec 2013)

The effective address of the operand is the contents of a register or memory location whose address appears in the instruction. Example **Add (R2),R0**

Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2. The initialization section of the program loads the counter value n from memory location N into R1 and uses the Immediate addressing mode to place the address value NUM 1, which is the address of the first number in the list, into R2.

64. List the eight great ideas invented by computer architects. (Nov/Dec-2015)

- Design for Moore's Law
- Use abstraction to simplify design
- Make the common case fast
- Performance via parallelism
- Performance via pipelining
- Performance via prediction
- Hierarchy of memories
- Dependability via redundancy

65. Distinguish pipelining from parallelism. (N/D2015)

- Parallelism means we are using more hardware for the executing the desired task. In parallel computing more than one processor are running in parallel. There may be some dedicated hardware running in parallel for doing the specific task.
- Parallelism increases the performance but the area also increases.
- The pipelining is an implementation technique in which multiple instructions are overlapped in execution.
- In case of pipelining the performance and throughput increases at the cost of pipelining registers area.
- In pipelining there are different hazards like data hazards, control hazards etc.

66. Give the formula for CPU execution time for a program.(Nov/Dec 2016)

- A simple formula relates the most basic metrics (clock cycles and clock cycle time) to CPU time:

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock cycle time}} \times \text{Clock cycle time}$$

- Alternatively, because clock rate and clock cycle time are inverses,

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

- This formula makes it clear that the hardware designer can improve performance by reducing the number of clock cycles required for a program or the length of the clock cycle.

67. What is an instruction register? (Nov/Dec 2016)

In computing, an instruction register (IR) is the part of a CPU's control unit that holds the instruction currently being executed or decoded.

68. State the need for indirect addressing mode. Give an example. Apr/May 2017

The register or memory location that contains the **address** of the operand is a pointer. When an execution takes place in such **mode**, instruction may be told to go to a specific **address**. Once it's there, instead of finding an operand, it finds an **address** where the operand is located.

In this case the number is usually enclosed with square brackets.

LD Acc, [5];Load the value stored in the memory location pointed to by the operand into the accumulator
Memory location 5 is accessed which contains 3. Memory location 3 is accessed which is 17.
Ax becomes 17.

69. Specify the CPU performance equation. Nov/ Dec 2012

The **performance equation** analyzes execution time as a product of three factors that are relatively independent of each other. The three factors are, in order, known as the instruction count (IC), clocks per instruction (CPI), and clock time (CT). CPI is computed as an effective value.

70. Write the equation for the dynamic power required per transistor. Apr. / May 2018

$$\text{Energy/transition} \quad E_{VDD} = \int_0^{\infty} iV_{DD}(t)V_{DD}dt = V_{DD} \int_0^{\infty} C_L \frac{dv_{out}}{dt} dt = C_L V_{DD} \int_0^{V_{DD}} dv_{out} = C_L V_{DD}^2$$

$$\text{Power} = \text{Energy/transition} * f = C_L * V_{dd}^2 * f$$

f represents the frequency of energy-consuming transitions (0→1)

71. Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5 GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and has a CPI of 2.2. Which processor has the highest performance expressed in instructions per second? Nov. / Dec. 2018

$$\text{Performance} = (\text{instructions/sec})$$

Processor	ClockRate	CPI	Performance
P1	3×10^9	1.5	$3 \times 10^9 / 1.5 = 2 \times 10^9$
P2	2.5×10^9	1.0	$2.5 \times 10^9 / 1.0 = 2.5 \times 10^9$
P3	4×10^9	2.2	$4 \times 10^9 / 2.2 = 1.8 \times 10^9$

P2 has the highest performance

72. Give the MIPS code for the statement $f=(g+h)-(i+j)$. May 2019

Simple arithmetic expression, assignment

int f, g, h, i, j;

f = (g + h) - (i + j);

\$s0	(g + h) - (i + j)
\$s1	i + j
\$s2	h
\$s3	i
\$s4	j

Assume variables are assigned to \$s0, \$s1, \$s2, \$s3, \$s4 respectively

add \$s0, \$s1, \$s2 # \$s0 = g + h

add \$s1, \$s3, \$s4 # \$s1 = i + j

sub \$s0, \$s0, \$s1 # f = (g + h) - (i + j)

73. Define Word Length. Nov/Dec-2019

A word is a unit of data of a defined bit length that can be addressed and moved between storage and the computer processor. ... Typically, an instruction is a word in length, but some architectures support halfword and doubleword-length instructions

74. What is Zero address instruction format? Nov/Dec 2020

A zero-address instruction implies that the absolute address of the operand is held in a special register that is automatically incremented (or decremented) to point to the location of the top of the stack.

75. What is the impact of frequency of clock signal applied to the microprocessor in the performance of computer? Nov/Dec 2020.

A computer's processor clock speed determines how quickly the central processing unit (CPU) can retrieve and interpret instructions. This helps your computer complete more tasks by getting them done faster. Clock speeds are measured in gigahertz (GHz), with a higher number equating to higher clock speed.

76. List the difference between wall clock time and response time. Nov/Dec 2021

Wall clock time is the actual amount of time taken to perform a job. This is equivalent to timing your job with a stopwatch and the measured time to complete your task can be affected by anything else that the system happens to be doing at the time.

User time measures the amount of time the CPU spent running your code. This does not count anything else that might be running, and also does not count CPU time spent in the kernel (such as for file I/O).

CPU time measures the total amount of time the CPU spent running your code or anything requested by your code. This includes kernel time.

The "User time" measurement is probably the most appropriate for measuring the performance of different jobs, since it will be least affected by other things happening on the system.

77. Find the Cycle time of a 450MHz clock frequency. Nov/Dec 2021

$$\text{Time} = 1/\text{frequency}$$

$$= 1/450 * 10^6$$

$$= 0.00222 * 10^{-6}$$

UNIT IV PROCESSOR

Instruction Execution – Building a Data Path – Designing a Control Unit – Hardwired Control, Micro programmed Control – Pipelining – Data Hazard – Control Hazards.

PART B

1. Briefly explain about Basic MIPS Implementation. Nov / Dec 2015, 2018

A Basic MIPS Implementation:

We will be examining an implementation that includes a subset of the core **MIPS instruction set:(Micro Instruction per Second)**

- **The memory**-reference instructions load word (lw) and store word (sw)
 - **The arithmetic**-logical instructions add, sub, AND, OR, and slt
 - **The instructions branch** equal (beq) and jump (j), which we add last
- This subset does not include all the integer instructions (for example, shift, multiply, and divide are issuing), nor does it include any floating-point instructions.
 - However, the key principles used in creating a data path and designing the control are illustrated.
 - The implementation of the remaining instructions is similar. In examining the implementation, we will have the opportunity to see how the instruction set architecture determines many aspects of the implementation, and how the choice of various implementation strategies affects the clock rate and CPI for the computer.
 - In addition, most concepts used to implement the MIPS subset in this chapter are the same basic ideas that are used to construct a broad spectrum of computers, from high-performance servers to general-purpose microprocessors to embedded processors.

An Overview of the Implementation

- MIPS instructions, including the integer arithmetic-logical instructions, the memory-reference instructions, and the branch instructions.
- What needs to be done to implement these instructions is the same, independent of the exact class of instruction.

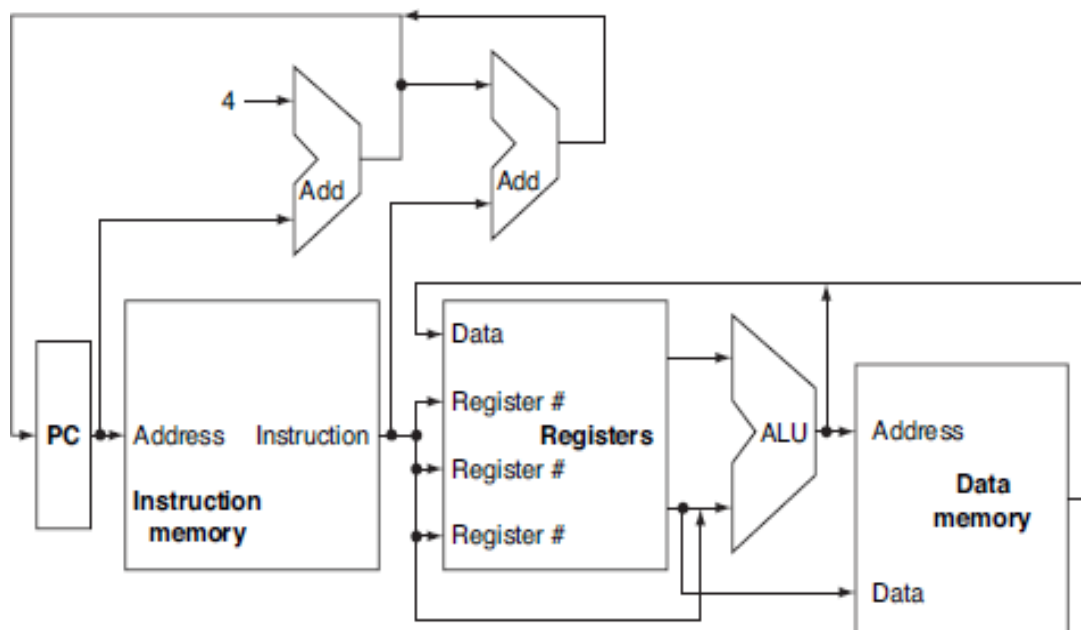
For every instruction, the first two steps are identical:

1. Send the program counter (PC) to the memory that contains the code and fetch the instruction from that memory.
2. Read one or two registers, using fields of the instruction to select the registers to read. For the load word instruction, we need to read only one register, but most other instructions require that we read two registers.

- After these two steps, the actions required to complete the instruction depend on the instruction class. Fortunately, for each of the three instruction classes (memory-reference, arithmetic-logical, and branches), the actions are largely the same, independent of the exact instruction.
- The simplicity and regularity of the MIPS instruction set simplifies the implementation by making the execution of many of the instruction classes similar.

For example,

- All instruction classes, except jump, use the arithmetic-logical unit (ALU) after reading the registers.
- The memory-reference instructions use the ALU for an address calculation, the arithmetic-logical instructions for the operation execution, and branches for comparison. After using the ALU, the actions required to complete various instruction classes differ.
- A memory-reference instruction will need to access the memory either to read data for a load or write data for a store.
- An arithmetic-logical or load instruction must write the data from the ALU or memory back into a register. Lastly, for a branch instruction, we may need to change the next instruction address based on the comparison; otherwise, the PC should be incremented by 4 to get the address of the next instruction.



An abstract view of the implementation of the MIPS subset showing the Major functional units and the major connections between them

- All instructions start by using the program counter to supply the instruction address to the instruction memory.
- After the instruction is fetched, the register operands used by an instruction are specified by fields of that instruction.

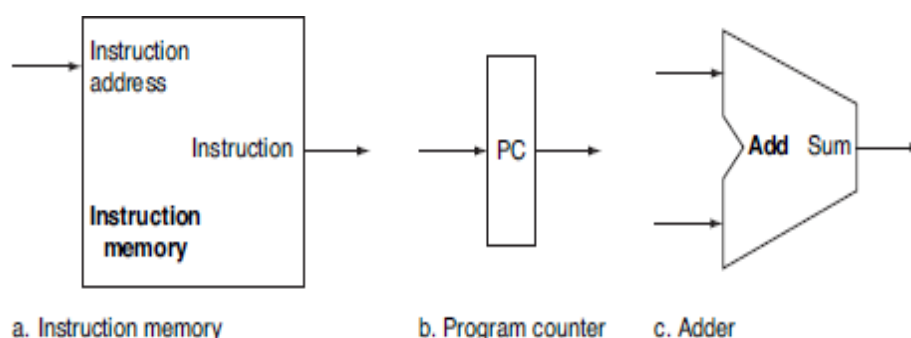
- Once the register operands have been fetched, they can be operated on to compute a memory address (for a load or store), to compute an arithmetic result (for an integer arithmetic-logical instruction), or a compare (for a branch).
- If the instruction is an arithmetic-logical instruction, the result from the ALU must be written to a register. If the operation is a load or store, the ALU result is used as an address to either store a value from the registers or load a value from memory into the registers.
- The result from the ALU or memory is written back into the register file. Branches require the use of the ALU output to determine the next instruction address, which comes either from the ALU (where the PC and branch offset are summed) or from an adder that increments the current PC by 4.
- The thick lines interconnecting the functional units represent buses, which consist of multiple signals. The arrows are used to guide the reader in knowing how information flows. Since signal lines may cross, we explicitly show when crossing lines are connected by the presence of a dot where the lines cross.

BUILDING A DATA PATH

2. Give detail description about Building a Data path.(or) Build a suitable Data path for branch instruction. Explain all the blocks with suitable example. Nov/Dec 2021

Data path element: A unit used to operate on or hold data within a processor. In the MIPS implementation, the data path elements include the instruction and data memories, the register file, the ALU and adders.

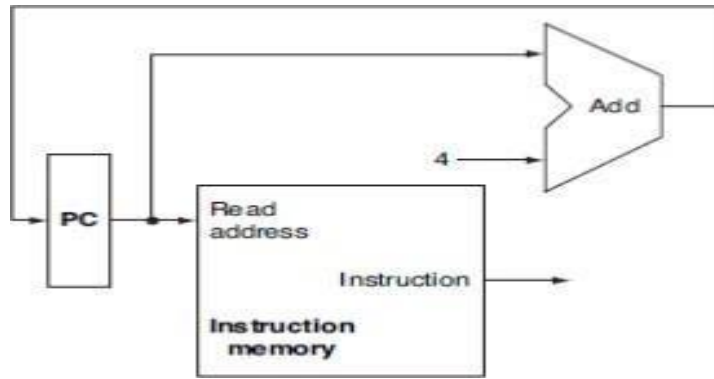
A memory unit to store the instructions of a program and supply instructions given an address. The program counter (PC), is a register that holds the address of the current instruction. We need an adder to increment the PC to the address of the next instruction.



Two state elements are needed to store and access instructions, and an adder is needed to compute the next instruction address.

- The state elements are the instruction memory and the program counter. The instruction memory need only provide read access because the data path does not write instructions.

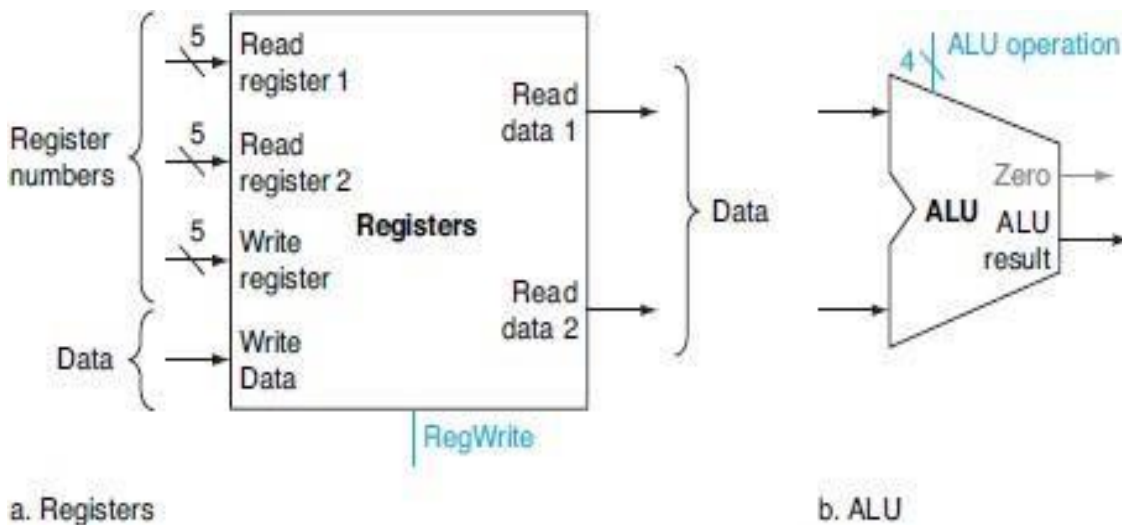
- Since the instruction memory only reads, we treat it as combinational logic: the output at any time reflects the contents of the location specified by the address input, and no read control signal is needed. (We will need to write the instruction memory when we load the program; this is not hard to add, and we ignore it for simplicity.)
- The program counter is a 32-bit register that is written at the end of every clock cycle and thus does not need a write control signal. The adder is an ALU wired to always add its two 32-bit inputs and place the sum on its output.
- Simply by wiring the control lines so that the control always specifies an add operation. We will draw such an ALU with the label *Add*, to indicate that it has been permanently made an adder and cannot perform the other ALU functions. To execute any instruction, we must start by fetching the instruction from memory.
- To prepare for executing the next instruction, we must also increment the program counter so that it points at the next instruction, 4 bytes later how to combine the three elements to form a datapath that fetches instructions and increments the PC to obtain the address of the next sequential instruction.
- Now let's consider the R-format instructions. They all read two registers, perform an ALU operation on the contents of the registers, and write the result to a register.
- We call these instructions either *R-type instructions* or *arithmetic-logical instructions* (since they perform arithmetic or logical operations). This instruction class includes **add, sub, AND, OR, and slt**, Recall that a typical instance of such an instruction is `add $t1,$t2,$t3`, which reads \$t2 and \$t3 and writes \$t1.
- The processor's 32 general-purpose registers are stored in a structure called a **register file**. A register file is a **collection of registers** in which any register can be read or written by specifying the number of the register in the file. The register file contains the register state of the computer.
- In addition, we will need an ALU to operate on the values read from the registers.
- R-format instructions have three register operands, so we will need to read two data words from the register file and write one data word into the register file for each instruction. For each data word to be read from the registers, input to the register file that specifies the register number to be read and an output from the register file that will carry the value that has been read from the registers.



A portion of the datapath used for fetching instructions and incrementing the program counter. The fetched instruction is used by other parts of the datapath.

To write a data word, we will need two inputs:

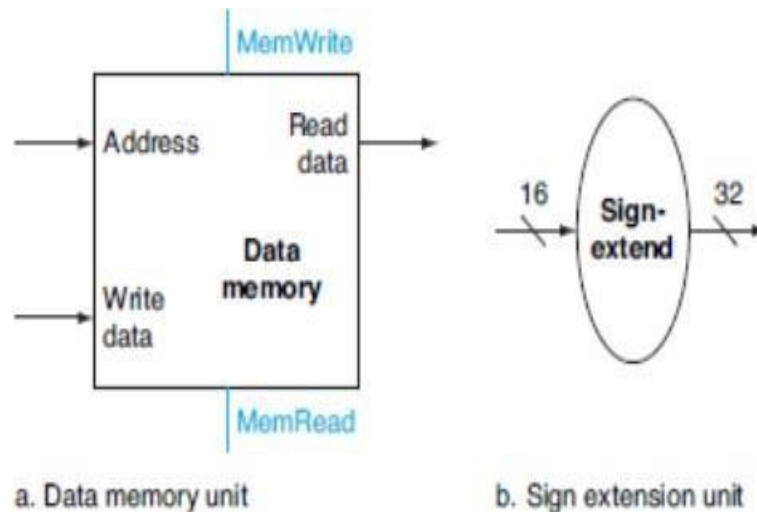
- One to specify the *register number* to be written and one to supply the *data* to be written into the register.
- The register file always outputs the contents of whatever register numbers are on the Read register inputs. Writes, however, are controlled by the write control signal, which must be asserted for a write to occur at the clock edge. We need a total of four inputs (**three for register numbers** and one for **data**) and two outputs (both for data). The register number inputs are 5 bits wide to specify one of 32 registers ($32 = 2^5$), whereas the data input and two data output buses are each 32 bits wide.



Register and ALU

- The ALU, which takes two 32-bit inputs and produces a 32-bit result, as well as a 1-bit signal if the result is 0. The 4-bit control signal of the ALU.
- **Sign-extend** to increase the size of a data item by replicating the high-order sign bit of the original data item in the high order bits of the larger, destination data item.
- **Sign-extend** the 16-bit offset field in the instruction to a 32-bit signed value, and a data memory unit to read from or write to. The data memory must be written on store instructions; hence, data

memory has read and writes control signals, an address input, and an input for the data to be written into memory.



The two units needed to implement loads and stores, in addition to the register file and ALU

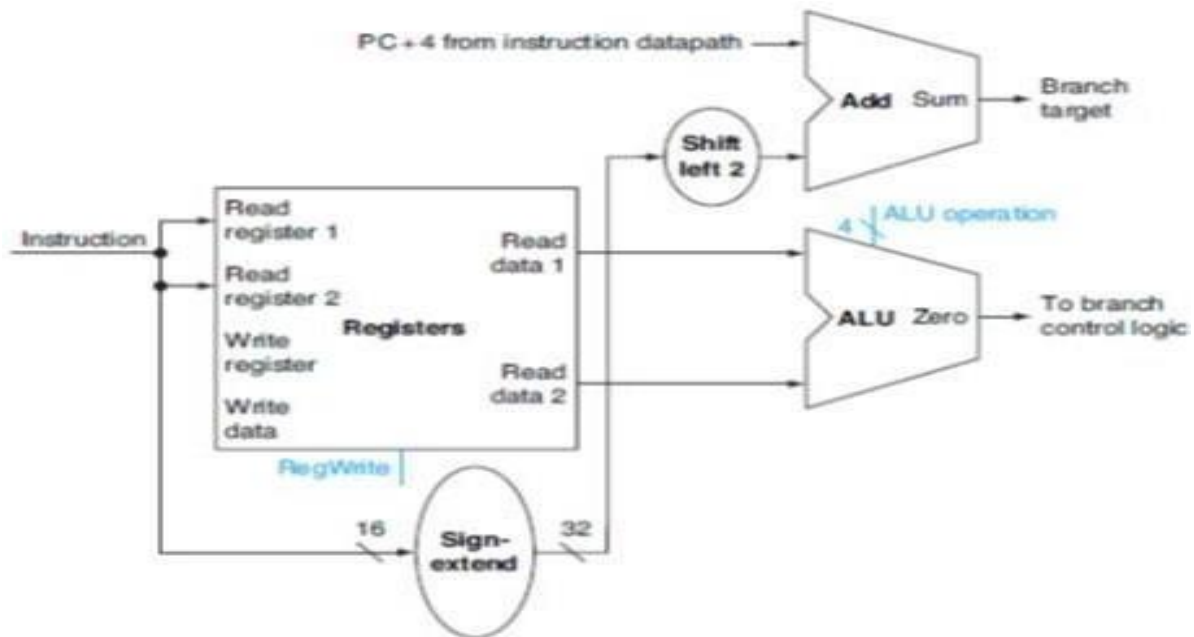
- The beq instruction has three operands, two registers that are compared for equality, and a 16-bit offset used to compute the **branch target address** relative to the branch instruction address. Its form is beq \$t1,\$t2,offset. To implement this instruction, we must compute the branch target address by adding the sign-extended offset field of the instruction to the PC.

There are two details in the definition of branch instructions

- The instruction set architecture specifies that the base for the branch address calculation is the address of the instruction following the branch. Since we compute PC + 4 (the address of the next instruction) in the instruction fetch datapath, it is easy to use this value as the base for computing the branch target address.
- The architecture also states that the offset field is shifted left 2 bits so that it is a word offset; this shift increases the effective range of the offset field by a factor of 4.

To deal with the later complication, we will need to shift the offset field by 2.

- **Branch taken.** A branch where the branch condition **is satisfied** and the program counter (PC) becomes the branch target. All unconditional branches are taken branches.
- **Branch not taken or (untaken branch)** .A branch where the branch condition **is false** and the program counter (PC) becomes the address of the instruction that sequentially follows the branch



The datapath for a branch uses the ALU to evaluate the branch condition and a separate adder to compute the branch target as the sum of the incremented PC and the sign-extended, lower 16 bits of the instruction (the branch displacement), shifted left 2 bits.

- The unit labeled *Shift left 2* is simply a routing of the signals between input and output that adds 00_{two} to the low-order end of the sign-extended offset field; no actual shift hardware is needed, since the amount of the “shift” is constant.
- Since we know that the offset was sign-extended from 16 bits, the shift will throw away only “sign bits.” Control logic is used to decide whether the incremented PC or branch target should replace the PC, based on the Zero output of the ALU.

DESIGNING A CONTROL UNIT

3. Briefly explain about Control Implementation scheme.

Control Implementation scheme:

Over view:

- The ALU Control:
- Designing the Main Control Unit
- Operation of the Datapath

The ALU Control:

The MIPS ALU defines the 6 following combinations of four control inputs:

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

Depending on the instruction class, the ALU will need to perform **one of these first five functions**.

(NOR is needed for other parts of the MIPS instruction set not found in the subset we are implementing.)

- For load word and store word instructions, we use the ALU to compute the memory address by addition.
- For the R-type instructions, the ALU needs to perform one of the five actions (AND, OR, subtract, add, or set on less than), depending on the value of the 6-bit funct (or function) field in the low-order bits of the instruction.
- For branch equal, the ALU must perform a subtraction.
- We can generate the 4-bit ALU control input using a small control unit that has inputs the function field of the instruction and a 2-bit control field, which we call ALUOp.
- ALUOp indicates whether the operation to be performed should be add (00) for loads and stores, subtract (01) for beq, or determined by the operation encoded in the funct field (10). The output of the ALU control unit is a 4-bit signal that directly controls the ALU by generating one of the 4-bit combinations shown previously.
- The below table shows how to set the ALU control inputs based on the 2-bit ALUOp control and the 6-bit function code.

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

How the ALU control bits are set depends on the ALUOp control bits and the different function codes for the R-type instruction.

- The opcode, listed in the first column, determines the setting of the ALUOp bits. All the encodings are shown in binary.
- Notice that when the ALUOp code is 00 or 01, the desired ALU action does not depend on the function code field; in this case, we say that we “don’t care” about the value of the function code, and the funct field is shown as XXXXXX. When the ALUOp value is 10, then the function code is used to set the ALU control input.

- There are several different ways to implement the mapping from the 2-bit ALUOp field and the 6-bit funct field to the four ALU operation control bits.
- Because only a small number of the 64 possible values of the function field are of interest and the function field is used only when the ALUOp bits equal 10, we can use a small piece of logic that recognizes the subset of possible values and causes the correct setting of the ALU control bits.

As a step in designing this logic, it is useful to create a truth table for the interesting combinations of the function code field and the ALU Op bits. The below **truth table** shows how the 4-bit ALU control is set depending on these **two input fields**.

ALUOp		Funct field						Operation
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	
0	0	X	X	X	X	X	X	0010
0	1	X	X	X	X	X	X	0110
1	0	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	0	X	X	0	1	0	0	0000
1	0	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

The truth table for the 4 ALU control bits (called Operation).

- The inputs are the **ALUOp** and **function code field**. Only the entries for which the ALU control is asserted are shown.
- Some don't-care entries have been added. For example, the ALUOp does not use the encoding 11, so the truth table can contain entries 1X and X1, rather than 10 and 01.
- Note that when the function field is used, the first 2 bits (F5 and F4) of these instructions are always 10, so they are **don't-care terms** and are replaced with XX in the truth table.
- **Don't-care term:** An element of a logical function in which the output does not depend on the values of all the inputs.

4. Give detail description about the Design of Main Control Unit.

Designing the Main Control Unit

To understand how to connect the fields of an instruction to the data path, it is useful to review the formats of the **three instruction classes**:

- The R-type instruction classes,
- Branch instruction classes, and
- Load-store instruction classes

Field	0	rs	rt	rd	shamt	funct
Bit positions	31:26	25:21	20:16	15:11	10:6	5:0

a. R-type instruction

Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

b. Load or store instruction

Field	4	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

c. Branch instruction

The three instruction classes (R-type, load and store, and branch) use two different instruction formats

The jump instructions use another format, which we will discuss shortly.

(a). Instruction format for R-format instructions, which all have an opcode of 0. These instructions have three register operands: rs, rt, and rd. Fields rs and rt are sources, and rd is the destination. The ALU function is in the funct field and is decoded by the ALU control design in the previous section. The R-type instructions that we implement are add, sub, AND, OR, and slt. The shamt field is used only for shifts; we will ignore it in this chapter.

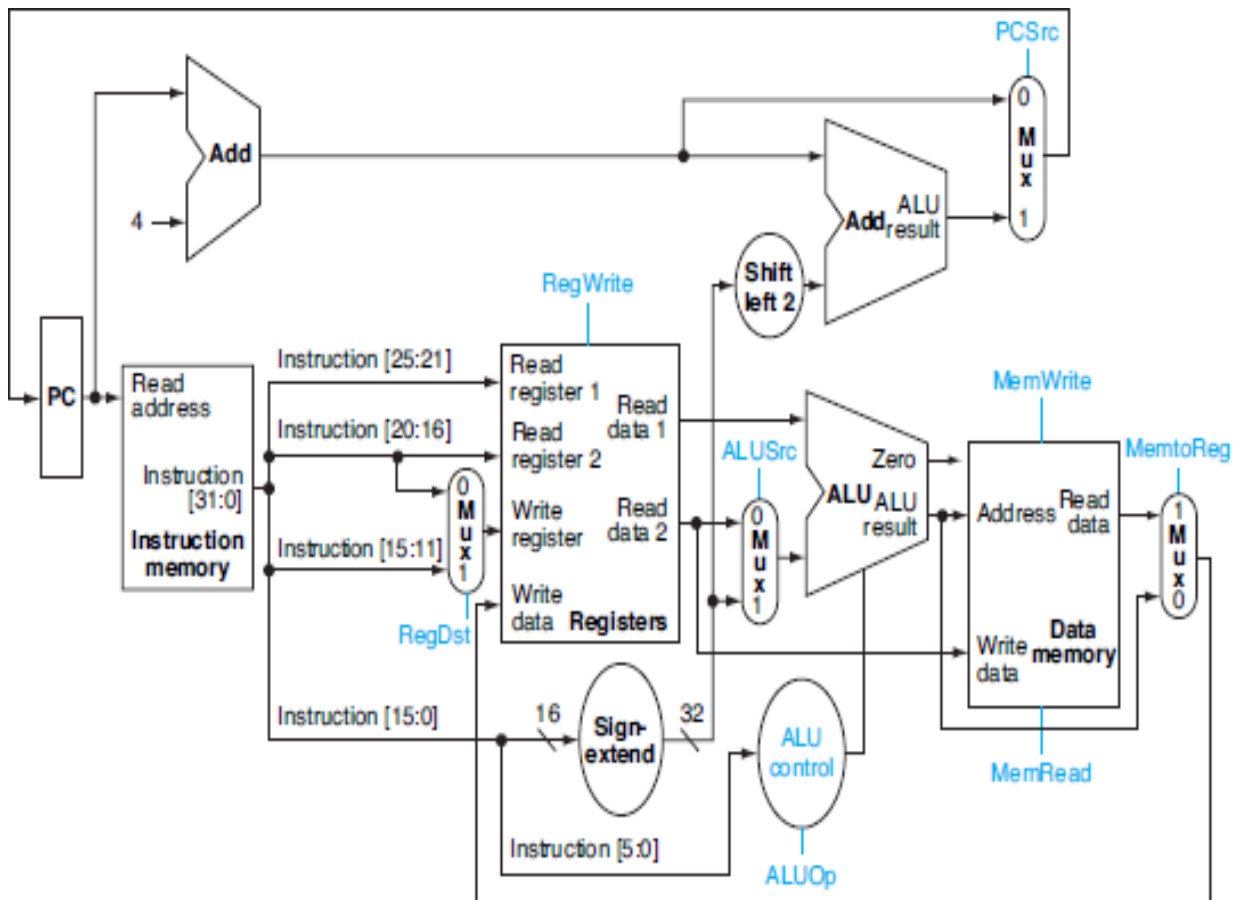
(b). Instruction format for load (opcode = 35_{ten}) and store (opcode = 43_{ten}) instructions. The register rs is the base register that is added to the 16-bit address field to form the memory address. For loads, rt is the destination register for the loaded value. For stores, rt is the source register whose value should be stored into memory.

(c). Instruction format for branch equal (opcode = 4). The registers rs and rt are the source registers that are compared for equality. The 16-bit address field is sign-extended, shifted, and added to the PC+4 to compute the branch target address.

There are several major observations about this instruction format that we will rely on:

- The op field, also called the **opcode**, is always contained in bits 31:26. We will refer to this field as Op[5:0].
- The two registers to be read are always specified by the rs and rt fields, at positions 25:21 and 20:16. This is true for the R-type instructions, branch equal, and store.
- The base register for load and store instructions is always in bit positions 25:21 (rs).
- The 16-bit offset for branch equal, load, and store is always in positions 15:0.
- The destination register is in one of two places. For a load it is in bit positions 20:16 (rt), while for an R-type instruction it is in bit positions 15:11 (rd). Thus, we will need to add a multiplexor to select which field of the instruction is used to indicate the register number to be written.

- Using this information, we can add the instruction labels and extra multiplexor (the Write register number input of the register file) to the simple datapath.
- These additions plus the ALU control block, the write signals for state elements, the read signal for the data memory, and the control signals for the multiplexors. Since all the multiplexors have two inputs, they each require a single control line.
- Seven single bit control lines plus the 2-bit ALUOp control signal. We have already defined how the ALUOp control signal works, and it is useful to define what the seven other control signals do informally before we determine how to set these control signals during instruction execution.



The data path of all necessary multiplexors and all control lines identified.

- The control lines are shown in color. The ALU control block has also been added. The PC does not require a write control, since it is written once at the end of every clock cycle; the branch control logic determines whether it is written with the incremented PC or the branch target address.

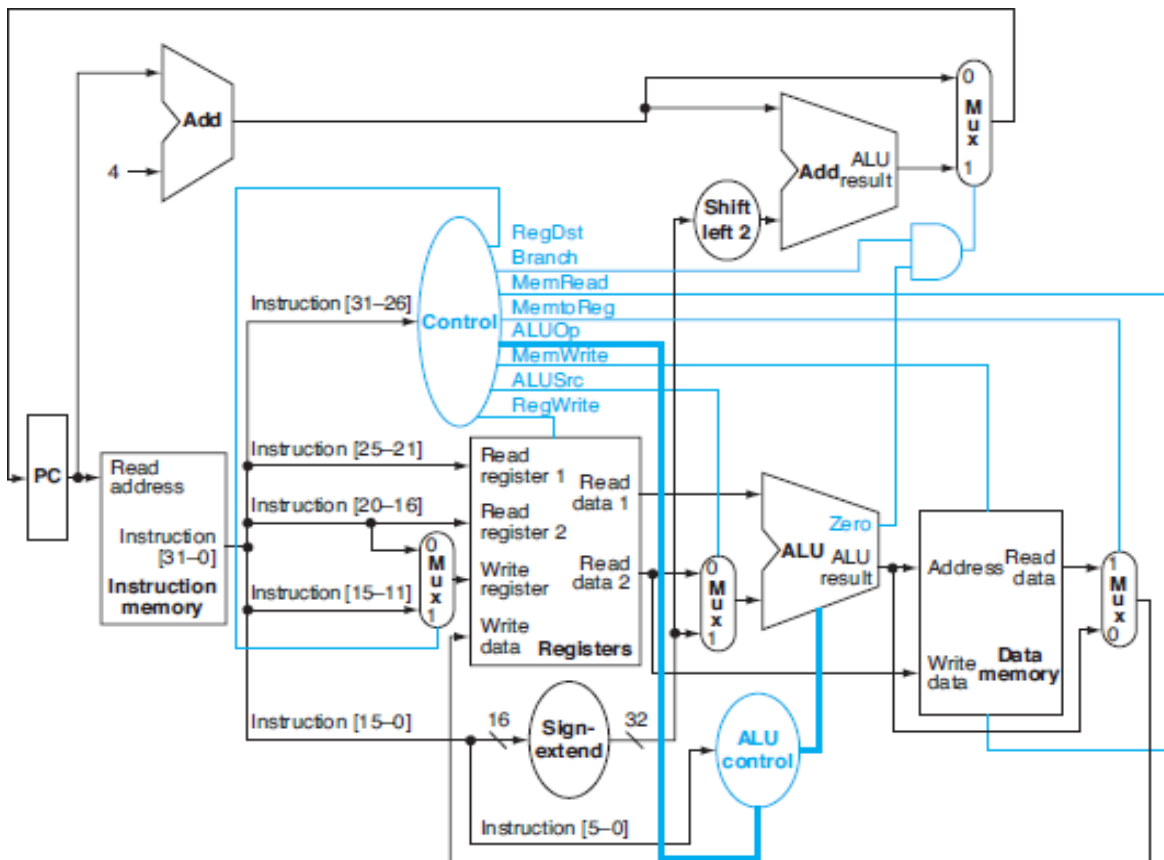
Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register Input is written with the value on the Write data Input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

The effect of each of the seven control signals.

- When the 1-bit control to a two-way multiplexor is asserted, the multiplexor selects the input corresponding to 1. Otherwise, if the control is disserted, the multiplexor selects the 0 input.
- Remember that the state elements all have the clock as an implicit input and that the clock is used in controlling writes. Gating the clock externally to a state element can create timing problems.

5. Briefly explain about Operation of the Data path with neat diagram. Apr. / May 2018,Nov/Dec2020. Nov/Dec 2021

Operation of the Data path:



The simple datapath with the control unit.

- The input to the control unit is the 6-bit opcode field from the instruction.
- The outputs of the control unit consist of three 1-bit signals that are used to control multiplexers (RegDst, ALUSrc, and MemtoReg), three signals for controlling reads and writes in the register file and data memory (RegWrite, MemRead, and MemWrite), a 1-bit signal used in determining whether to possibly branch (Branch), and a 2-bit control signal for the ALU (ALUOp).
- An AND gate is used to combine the branch control signal and the Zero output from the ALU.
- The AND gate output controls the selection of the next PC. Notice that PCSrc is now a derived signal, rather than one coming directly from the control unit. Thus, we drop the signal name in subsequent figures.
- The operation of the datapath for an R-type instruction, such as `add $t1,$t2,$t3`. Although everything occurs in one clock cycle, we can think of four steps to execute the instruction;

These steps are ordered by the flow of information:

1. The instruction is fetched, and the PC is incremented.
2. Two registers, \$t2 and \$t3, are read from the register file; also, the main control unit computes the setting of the control lines during this step.
3. The ALU operates on the data read from the register file, using the function code (bits 5:0, which is the funct field, of the instruction) to generate the ALU function.
4. The result from the ALU is written into the register file using bits 15:11 of the instruction to select the destination register (\$t1). Similarly, we can illustrate the execution of a load word, such as

```
lw $t1, offset($t2)
```

The active functional units and asserted control lines for a load. We can think of a load instruction as operating in

Five steps (similar to the R-type executed in four):

1. An instruction is fetched from the instruction memory, and the PC is incremented.
2. A register (\$t2) value is read from the register file.
3. The ALU computes the sum of the value read from the register file and the Sign-extended, lower 16 bits of the instruction (offset).
4. The sum from the ALU is used as the address for the data memory.
5. The data from the memory unit is written into the register file; the register destination is given by bits 20:16 of the instruction (\$t1).

The data path in operation for a load instruction:

- The control lines, data path units, and connections that are active are highlighted.
- A store instruction would operate very similarly. The main difference would be that the memory control would indicate a write rather than a read, the second register value read would be used

for the data to store, and the operation of writing the data memory value to the register file would not occur.

The data path in operation for a branch-on-equal instruction.

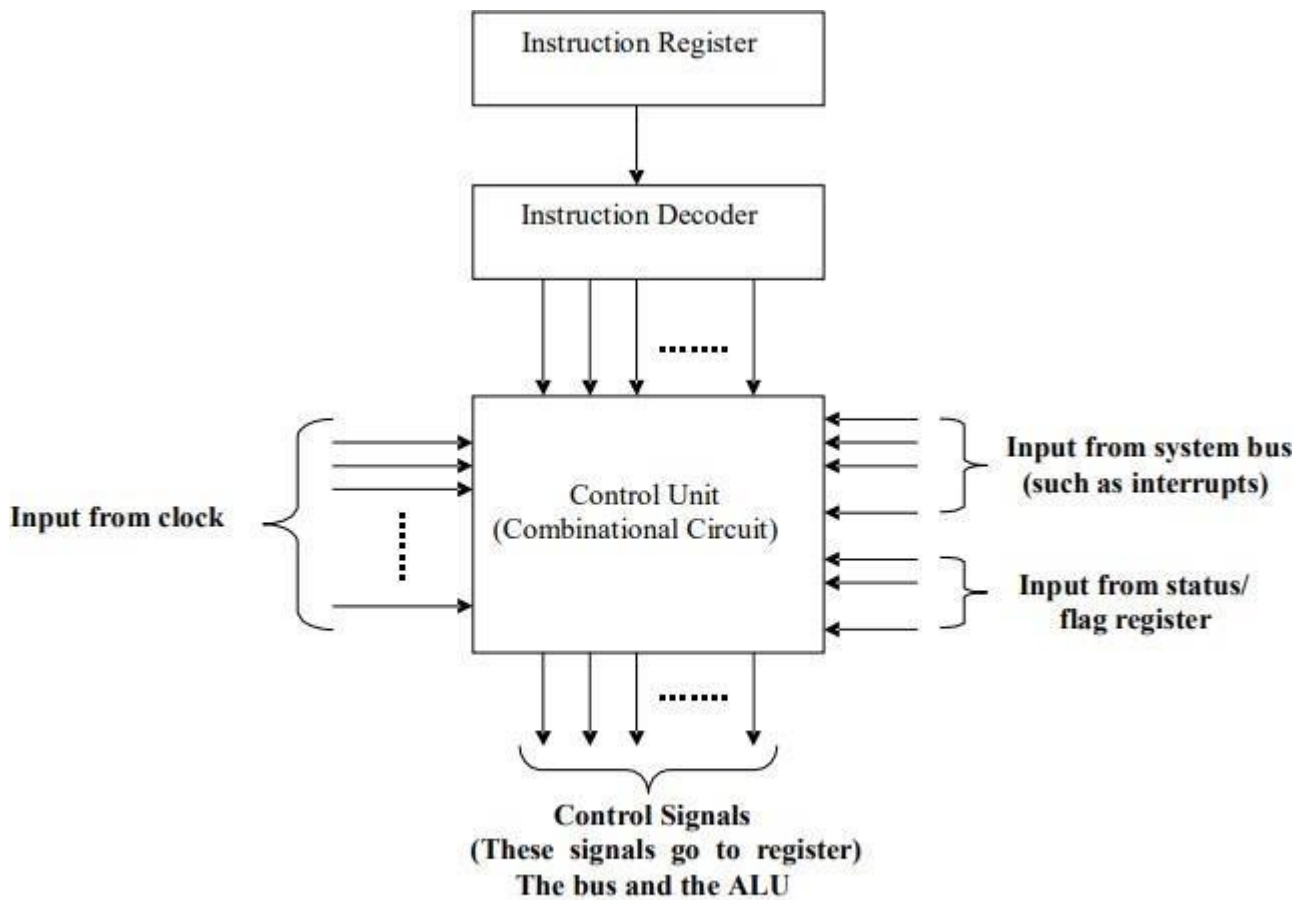
- Finally, we can show the operation of the branch-on-equal instruction, such as **beq \$t1,\$t2,offset** in the same fashion.
- It operates much like an R-format instruction, but the ALU output is used to determine whether the PC is written with PC + 4 or the branch target address.

The four steps for execution:

1. An instruction is fetched from the instruction memory, and the PC is incremented.
2. Two registers, \$t1 and \$t2, are read from the register file.
3. The ALU performs subtract operation on the data values read from the register file. The value of PC+4 is added to the sign-extended, lower 16 bits of the instruction (offset) shifted left by two; Result is in the branch target address.
4. The Zero result from the ALU is used to decide which address result to store into the PC.

HARDWIRED CONTROL AND MICRO PROGRAMMED CONTROL

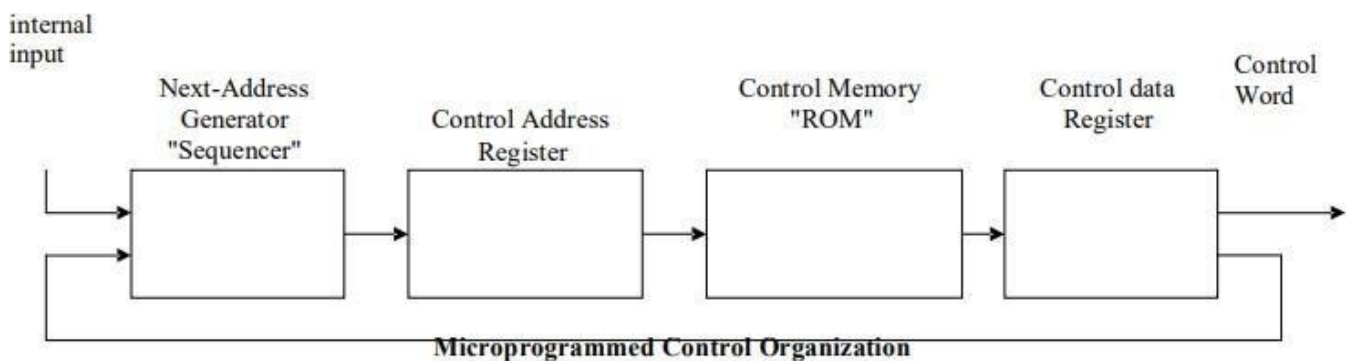
- Hardwired and Micro programmed Control For each instruction, the control unit causes the CPU to execute a sequence of steps correctly.
- In reality, there must be control signals to assert lines on various digital components to make things happen.
- For example, when we perform an Add instruction in assembly language, we assume the addition takes place because the control signals for the ALU are set to "add" and the result is put into the AC.
- The ALU has various control lines that determine which operation to perform. The question we need to answer is, "How do these control lines actually become asserted?" We can take one of two approaches to ensure control lines are set properly.
- The first approach is to physically connect all of the control lines to the actual machine instructions. The instructions are divided up into fields, and different bits in the instruction are combined through various digital logic components to drive the control lines.
- This is called hardwired control, and is illustrated in figure



Hardwired Control Organization

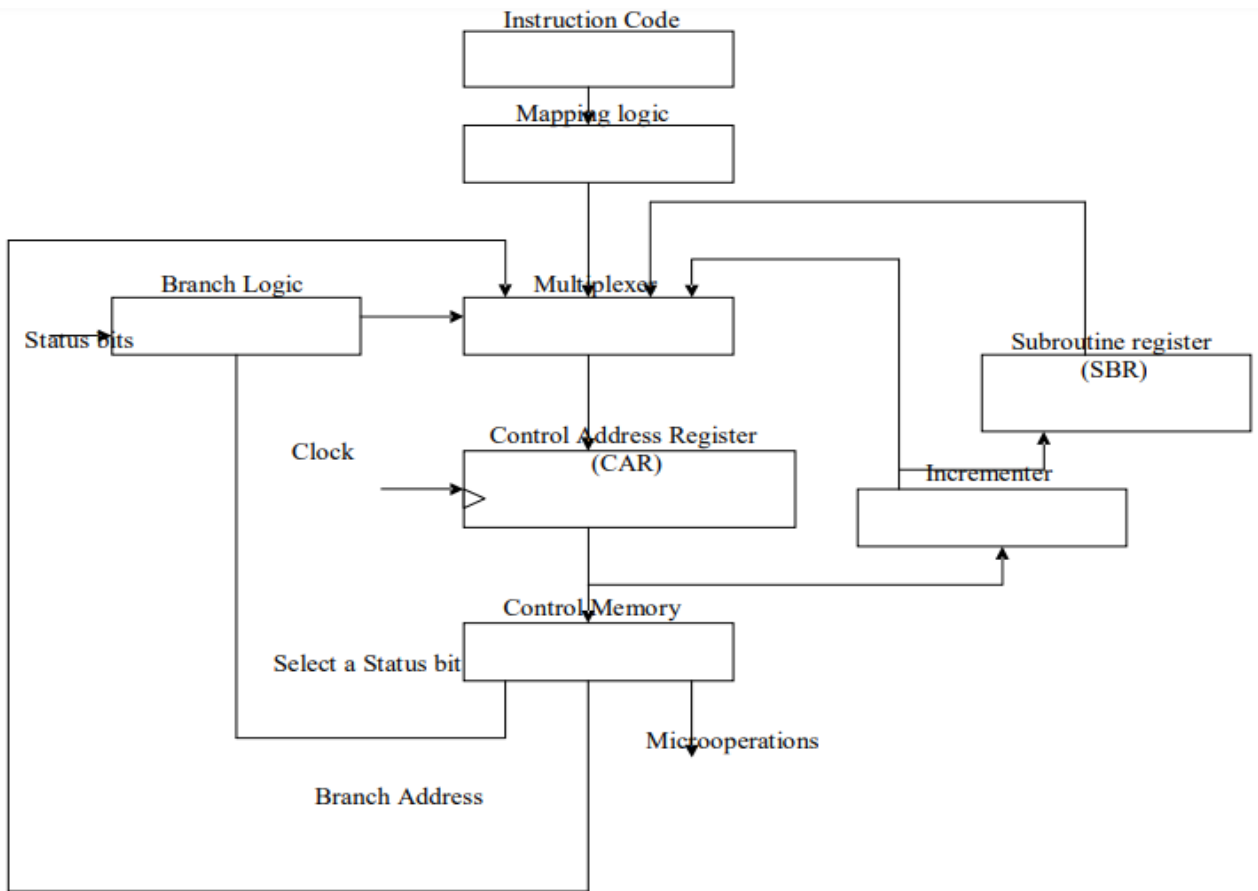
- The control unit is implemented using hardware (for example: NAND gates, flip-flops, and counters). We need a special digital circuit that uses, as inputs, the bits from the Opcode field in our instructions, bits from the flag (or status) register, signals from the bus, and signals from the clock.
- It should produce, as outputs, the control signals to drive the various components in the computer. The advantage of hardwired control is that it is very fast.
- The disadvantage is that the instruction set and the control logic are directly tied together by special circuits that are complex and difficult to design or modify.
- If someone designs a hardwired computer and later decides to extend the instruction set, the physical components in the computer must be changed.
- This is prohibitively expensive, because not only must new chips be fabricated but also the old ones must be located and replaced. Microprogramming is a second alternative for designing control unit of digital computer (uses software for control).
- A control unit whose binary control variables are stored in memory is called a micro programmed control unit. The control variables at any given time can be represented by a string of 1's and 0's called a control word (which can be programmed to perform various operations on the component of the system).

- Each word in control memory contains within it a microinstruction. The microinstruction specifies one or more micro operations for the system. A sequence of microinstructions constitutes a micro program.
- A memory that is part of a control unit is referred to as a control memory. A more advanced development known as dynamic microprogramming permits a micro program to be loaded initially from an auxiliary memory such as a magnetic disk.
- Control units that use dynamic microprogramming employ a writable control memory; this type of memory can be used for writing (to change the micro program) but is used mostly for reading.
- The general configuration of a micro programmed control unit is demonstrated in the block diagram of Figure.
- The control memory is assumed to be a ROM, within which all control information is permanently stored.



- The control memory address register specifies the address of the microinstruction and the control data register holds the microinstruction read from memory the microinstruction contains a control word that specifies one or more micro operations for the data processor.
- Once these operations are executed, the control must determine the next address. The location of the next microinstruction may be the one next in sequence, or it may be locate somewhere else in the control memory. For this reason it is necessary to use some bits of the present microinstruction to control the generation of the address of the next microinstruction.
- The next address may also be a function of external input conditions. While the micro operations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.
- The next address generator is sometimes called a micro program sequencer, as it determines the address sequence that is read from control memory, the address of the next microinstruction can be specified several ways, depending on the sequencer inputs.
- Typical functions of a micro program sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address or loading an initial address to start the control operations.

- The main advantages of the micro programmed control are the fact that once the hardware configuration is established; there should be no need for further hardware or wiring changes.
- If we want to establish are different control sequence for the system, all we need to do is specify different set microinstructions for control memory.
- The hardware configuration should not be changed for different operations; the only thing that must be changed is the micro program residing in control memory.
- Microinstructions are stored in control memory in groups, with each group specifying routine. Each computer instruction has micro program routine in control memory to generate the micro operations that execute the instruction.
- The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be to branch from one routine to another. The address sequencing capabilities required in a control memory are:
 1. Incrementing of the control address register.
 2. Unconditional branch or conditional branch, depending on status bit conditions.
 3. A mapping process from the bits of the instruction to an address for control memory.
 4. A facility for subroutine call and return.
- Figure shows a block diagram of control memory and the associated hardware needed for selecting the next microinstruction address.
- The microinstruction in control memory contains a set of bits to initiate micro operations in computer registers and other bits to specify the method by which the address is obtained.
- The diagram shows four different paths from which the control address register (CAR) receives the address.
- The incrementer increments the content of the control address register by one, to select the next microinstruction in sequence.
- Branching is achieved by specifying the branch address in one of the fields of the microinstruction.
- Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition.
- An external address is transferred into control memory via a mapping logic circuit. The return address for a subroutine is stored in a special register whose value is then used when the micro program wishes to return from the subroutine.



Selection address for control memory

PIPELINING

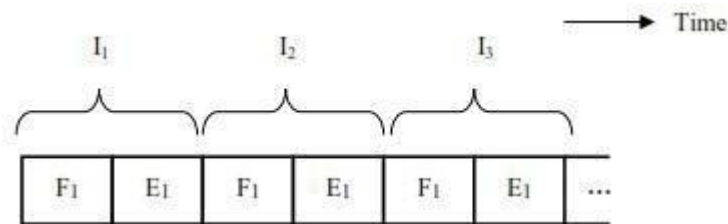
6. Explain a 4-stage instruction pipeline. Explain the issues affecting pipeline performance. (Or) Discuss the basic concepts of pipelining. (Apr/May2012) (May/June2013)Nov / Dec 2015, 2016,Nov/Dec 2020.

OVERVIEW:

- **Role of cache memory**
- **Pipelining Performance**

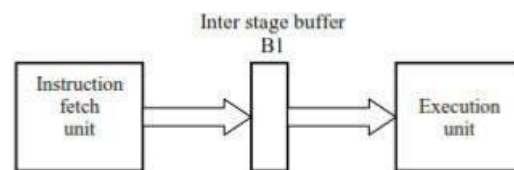
- Pipelining is an implementation technique in which multiple instructions are overlapped in execution pipelining is key to make processor fast.
- A pipeline can be visualized as a collection of processing segments through which binary information follows.
- In computer architecture Pipelining means executing machine instructions concurrently. The pipelining is used in modern computers to achieve high performance. The speed of execution of programs is influenced by **many factors**.
- One way to improve performance is to use faster circuit technology to build the processor and the main memory.
- Another possibility is to arrange the hardware so that more than one operation can be performed at the same time

- In this way, the number of operations performed per second is increased even though the elapsed time needed to perform any one operation is not changed.
- Pipelining is a particularly effective way of organizing concurrent activity in a computer system.
- The basic idea is very simple. It is frequently encountered in manufacturing plants, where pipelining is commonly known as an assembly-line operation.
- The processor executes a program by fetching and executing instructions, one after the other. Let F_i and E_i refer to the fetch and execute steps for instruction I_i .
- An execution of a program consists of a sequence of fetch and execute steps as shown below.



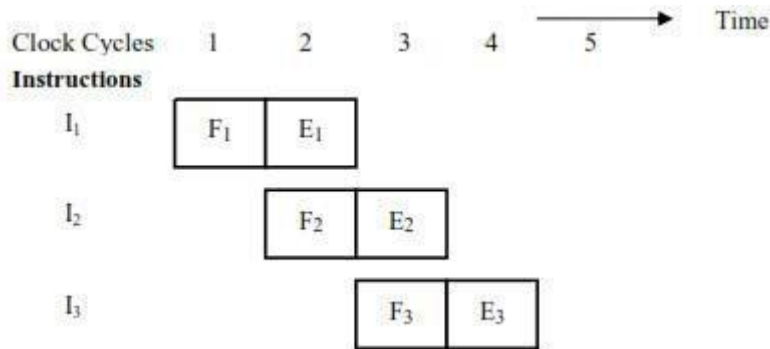
Sequential executions of instructions.

- Now consider a computer that has two separate hardware units, one for fetching instructions and another for executing them, as shown below. The instruction fetched by the fetch unit is deposited in an intermediate storage buffer B1.
- This buffer is needed to enable the execution unit to execute the instruction while the fetch unit is fetching the next instruction.
- The results of execution are deposited in the destination location specified by the instruction. The data can be operated by the instructions are inside the block labeled "**Execution unit**".



Hardware organization of pipelining.

- The computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can each be completed in one clock cycle.
- In the first clock cycle, the fetch unit fetches an instruction I_1 (step F_1) and stores it in buffer B1 at the end of the clock cycle.
- In the second clock cycle, the instruction fetch unit proceeds with the fetch operation for instruction I_2 (step F_2). Meanwhile, the execution unit performs the operation specified by instruction I_1 , which is available to it in buffer B1 (step E_1).
- By the end of the second clock cycle, the execution of instruction I_1 is completed and instruction I_2 is available. Instruction I_2 is stored in B1, replacing I_1 , which is no longer needed. Step E_2 is performed by the execution unit during the third clock cycle, while instruction I_3 is being fetched by the fetch unit. In this manner, both the fetch and execute units are kept busy all the time.

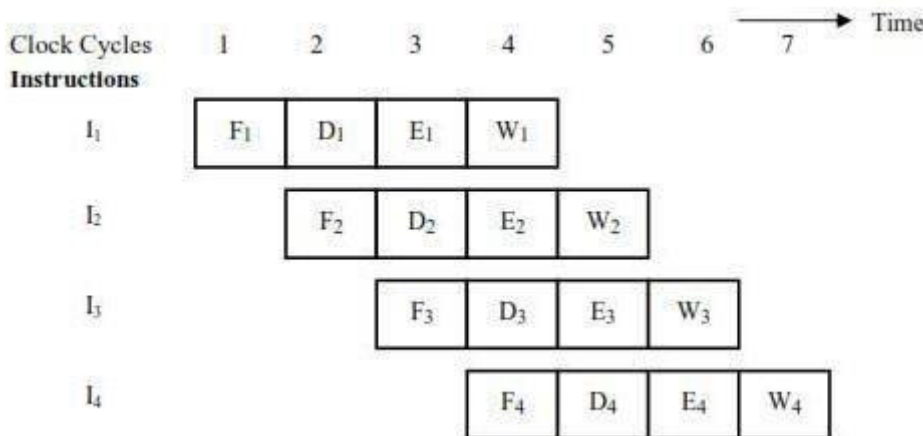


Pipelined executions of instructions (Instructions Pipelining)

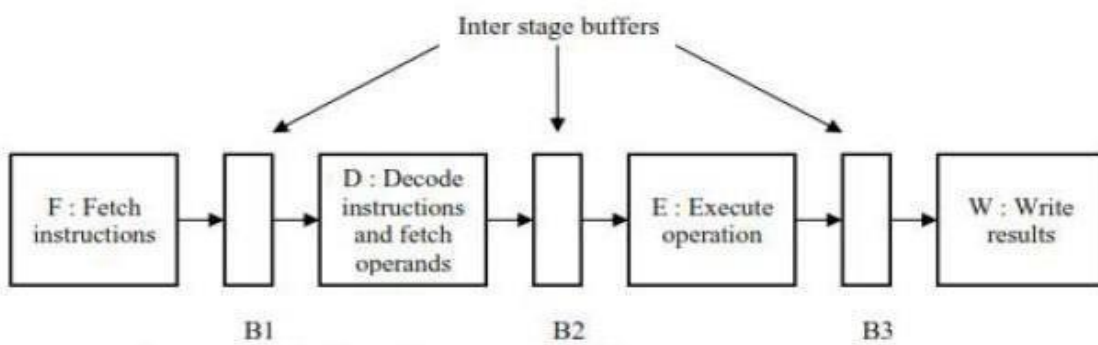
A pipelined processor may process each instruction in **four steps**, as follows:

- **F Fetch:** read the instruction from the memory.
- **D Decode:** decode the instruction and fetch the source operand(s).
- **E Execute:** perform the operation specified by the instruction.
- **W Write:** store the result in the destination location.

The sequence of events for this case is shown below. Four instructions are in progress at any given time. This means that four distinct hardware units are needed. These units must be capable of performing their tasks simultaneously and without interfering with one another. Information is passed from one unit to the next through a storage buffer.



Instruction execution divided into four steps.



Hardware organization of a 4- stage pipeline

For example, during clock cycle 4, the information in the buffers is as follows:

- Buffer B1 holds instruction I3, which was fetched in cycle 3 and is being decoded by the Instruction-decoding unit.
- Buffer B2 holds both the source operands for instruction I2 and the specification of the operation to be performed.
- Buffer B3 holds the results produced by the execution unit and the destination information for instruction I1.

ROLE OF CACHE MEMORY:

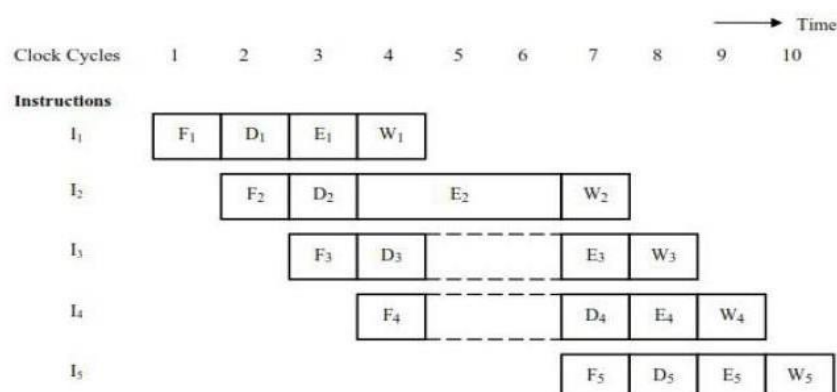
- Each stage in a pipeline is expected to complete its operation in one clock cycle. Hence, the clock period should be sufficiently long to complete the task being performed in any stage.
- Pipelining is most effective in **improving performance** if the tasks being performed in different stages require about the same amount of time.
- The use of cache memories solves the memory access problem. In particular, when a cache is included on the same chip as the processor, access time to the cache is usually the same as the time needed to perform other basic operations inside the processor.
- This makes it possible to divide instruction fetching and processing into steps that are more or less equal in duration. Each of these steps is performed by a different pipeline stage, and the clock period is chosen to correspond to the longest one.

7. Explain about Pipeline Performance. (Or) How to measure the performance of a pipeline? (Or) List the key aspects in gaining the performance in pipelined systems. (Apr/May 2010) or Explain the difference types of pipeline hazards with suitable examples. (16 Marks) Apr / May 2015, 2016, Nov. / Dec. 2018 (Nov/Dec 2019) Nov/Dec 2020. Nov/Dec 2021

PIPELINE PERFORMANCE:

The pipelined processor completes the processing of one instruction in each clock cycle, which means that the rate of instruction processing is four times that of sequential operation.

- The potential increase in performance resulting from pipelining is proportional to the number of pipeline stages.
- However, this increase would be achieved only if pipelined operation as depicted could be sustained without interruption throughout program execution. Unfortunately, this is not the case.



Effect of an execution operation taking more than one clock cycle

Performance measures:

The various performance measures of pipelining are

- Throughput
- CPI
- Speedup
- Dependencies
- Hazards

Throughput

- The number of instruction executed per secondCPI(clock cycle per instruction)
- The CPI and MIPS can be related by the equation

$$\text{CPI} = f / \text{MIPS}$$

Where F is the clock frequency in MHz

Speedup

- Speedup is defined by

$$S(m) = T(1) / T(m)$$

- Where T (m) is the execution time for some target workload on an m-stage pipeline and T(1) is the execution time for same target workload on a non-pipelined **Processor**.

Dependencies

- If the output of any stage interferes the execution of other stage then dependencies exists.

There are two types of dependencies. They are

1. **Control dependency**
2. **Data dependency**

**8. Give detail description about Pipelined data path and control. (Nov/Dec2014)(Apr/May2012)
(Or) Discuss the modified data path to accommodate pipelined executions with a diagram.
Apr/May 2016, 2017, 2018Nov/Dec 2021**

Pipelined data path and control

Consider the three-bus structure suitable for pipelined execution with a slight modification to support a 4-stage pipeline. Several important changes are

- There are *separate instruction and data caches* that use separate address and data connections to the processor. This requires two versions of the MAR register, IMAR for accessing tile instruction cache and DMAR for accessing the data cache.
- The PC is connected *directly* to the IMAR, so that the contents of the PC can be transferred to IMAR at the same time that an independent ALU operation is taking place.
- The data address in DMAR can be obtained *directly from the register file or from the ALU* to support the register indirect and indexed addressing modes.

- Separate MDR registers are provided for *read and write operations*. Data can be transferred directly between these registers and the register file during load and store operations without the need to pass through the ALU.
- *Buffer registers* have been introduced at the inputs and output of the ALU. These are registers SRC1, SRC2, and RSLT. Forwarding connections may be added if desired.
- The instruction register has been replaced with an *instruction queue*, which is loaded from the instruction cache.
- The output of the instruction decoder is connected to the *control signal pipeline*. This pipeline holds the control signals in buffers B2 and B3.

The following operations can be performed independently in the process,

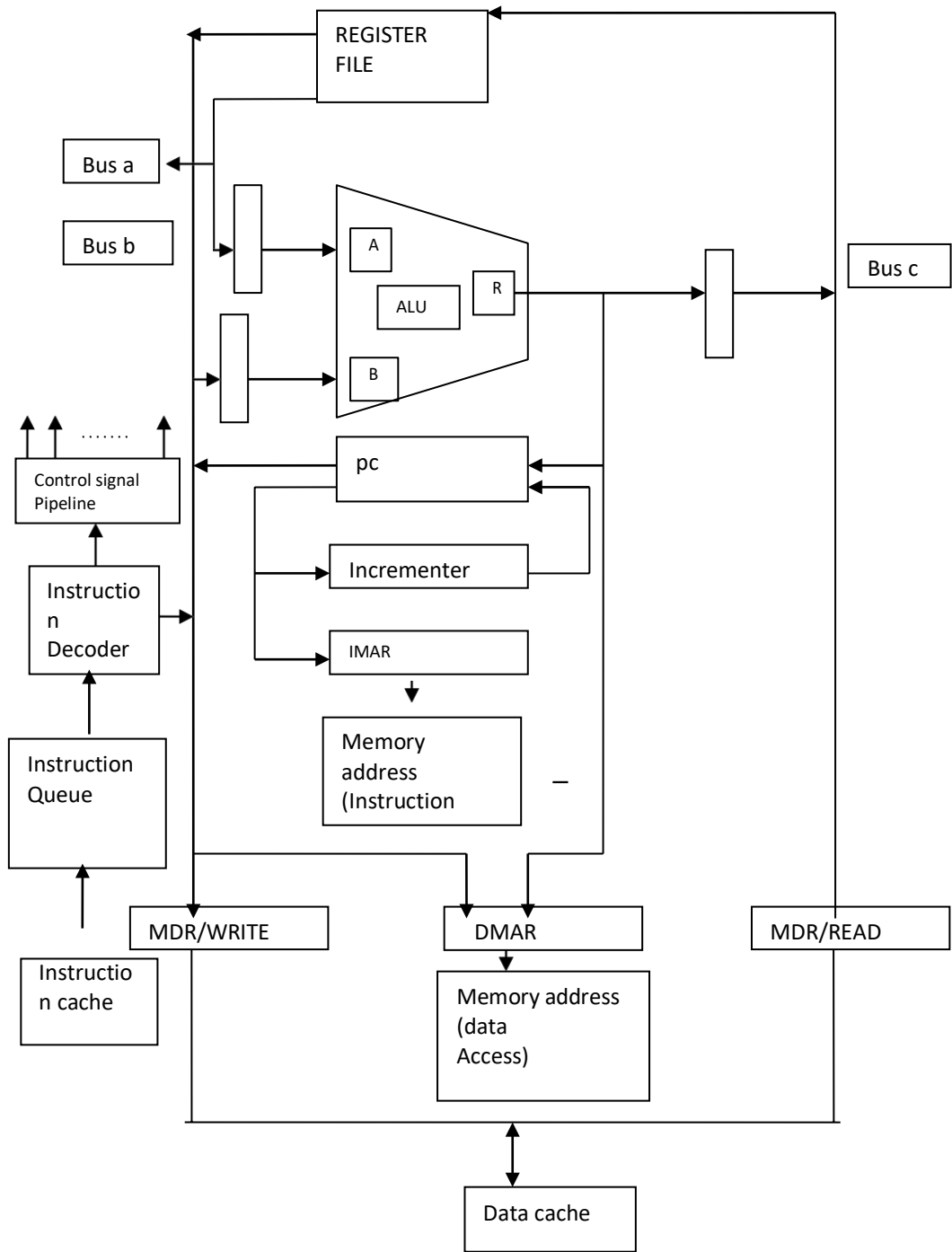
- Reading an instruction from the instruction cache
- Incrementing the pc
- Decoding the instruction
- Reading from or writing into the data cache.
- Reading the contents of up to two registers from the register file.
- Writing in to one register in the register file
- Performing an ALU operation.

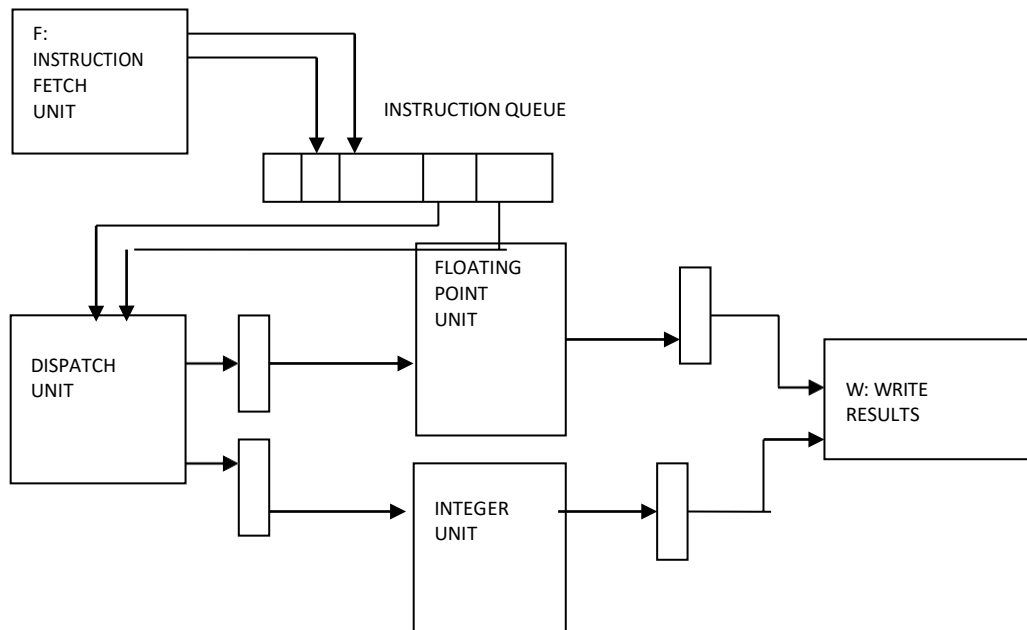
The structure provides the flexibility required to implement the four-stage pipeline.

For example: I1, I2, I3, I4

Be the sequence of four instructions.

- Write the result of instruction I1 into the register file.
- Read the operands of instruction I2 from the register file.
- Decode instruction I3
- Fetch instruction I4 and increment the PC.





Use of instruction queue in hardware organization

Advantages of Pipelining:

- The cycle time of the processor is reduced; increasing the instruction throughput. Pipelining doesn't reduce the time it takes to complete an instruction; instead it increases the number of instructions that can be processed simultaneously ("at once") and reduces the delay between completed instructions (called 'throughput'). The more pipeline stages a processor has, the more instructions it can process "at once" and the less of a delay there is between completed instructions. Every predominant general purpose microprocessor manufactured today uses at least 2 stages of pipeline up to 30 or 40 stages.
- If pipelining is used, the CPU Arithmetic logic unit can be designed faster, but more complex.
- Pipelining in theory increases performance over an un-pipelined core by a factor of the number of stages (assuming the clock frequency also increases by the same factor) and the code is ideal for pipeline execution.
- Pipelined CPUs generally work at a higher clock frequency than the RAM clock frequency, (as of 2008 technologies, RAMs work at a low frequencies compared to CPUs frequencies) increasing computers overall performance.

Disadvantages of Pipelining:

Pipelining has many disadvantages though there are a lot of techniques used by CPUs and compilers/designers to overcome most of them; the following is a list of common drawbacks:

- The design of a non-pipelined processor is simpler and cheaper to manufacture, non-pipelined processor executes only a single instruction at a time.
- This prevents branch delays (in Pipelining, every branch is delayed) as well as problems when serial instructions being executed concurrently.

- In pipelined processor, insertion of flip flops between modules increases the instruction latency compared to a non-pipelining processor.
- A non-pipelined processor will have a defined instruction throughput. The performance of a pipelined processor is much harder to predict and may vary widely for different programs.
- Many designs include pipelines as long as 7, 10, 20, 31 and even more stages; a disadvantage of a long pipeline is when a program branches, the entire pipeline must be flushed (cleared).
- The higher throughput of pipelines falls short when the executed code contains many branches: the processor cannot know in advance where to read the next instruction, and must wait for the branch instruction to finish, leaving the pipeline behind it empty.
- This disadvantage can be reduced by predicting whether the conditional branch instruction will branch based on previous activity.
- After the branch is resolved, the next instruction has to travel all the way through the pipeline before its result becomes available and the processor resumes "working" again.
- In such extreme cases, the performance of a pipelined processor could be worse than non-pipelined processor.
- Unfortunately, not all instructions are independent. In a simple pipeline, completing an instruction may require 5 stages. To operate at full performance, this pipeline will need to run 4 subsequent independent instructions while the first is completing.
- Any of those 4 instructions might depend on the output of the first instruction, causing the pipeline control logic to wait and insert a stall or wasted clock cycle into the pipeline until the dependency is resolved.
- Fortunately, techniques such as forwarding can significantly reduce the cases where stalling is required.
- Self-modifying programs may fail to execute properly on a pipelined architecture when the instructions being modified are near the instructions being executed.
- This can be caused by the instructions may already being in the Prefetch Input Queue, so the modification may not take effect for the upcoming execution of instructions. Instruction caches make the problem even worse.
- **Hazards:** When a programmer (or compiler) writes assembly code, they generally assume that each instruction is executed before the next instruction is being executed.
- When this assumption is not validated by pipelining it causes a program to behave incorrectly, the situation is known as a **hazard**. Various techniques for resolving hazards or working around such as forwarding and delaying (by inserting a stall or a wasted clock cycle) exist.

HAZARD

- Any condition that causes the pipeline to stall is called a *hazard*.

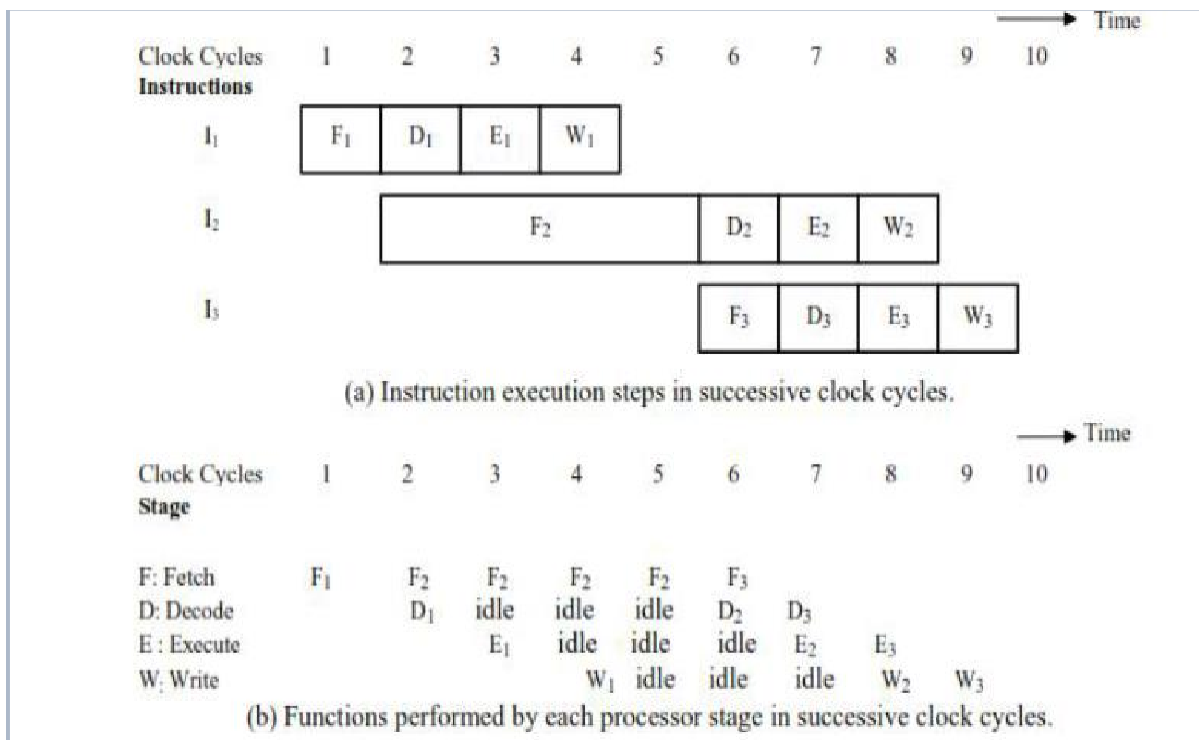
There are three type of hazard

I. Data Hazards

II. Control/Instruction hazards

III. Structural Hazard

- The operation specified in instruction I2 requires three cycles to complete, from cycle 4 through cycle 6. Thus, in cycles 5 and 6, the Write stage must be told to do nothing, because it has no data to work with. Meanwhile, the information in buffer B2 must remain intact until the Execute stage has completed its operation.
- This means that stage 2 and, in turn, stage1 are blocked from accepting new instructions because the information in B1 cannot be overwritten.
- Thus, steps D4 and F5 must be postponed as shown below. Pipelined operation is said to have been stalled for two clock cycles. Normal pipelined operation resumes in cycle 7.



Pipeline stall caused by a cache miss in F2

9. Briefly explain about how to handle the Data hazard.(Nov/Dec2012, 2014)(Apr/May2015) Or What is a data hazard? How do you overcome it? Discuss its side effects.(Apr/May 2014) Or Describe operand forwarding in a pipeline processor with a diagram. (6) Apr/ May 2017 Nov/Dec 2020.

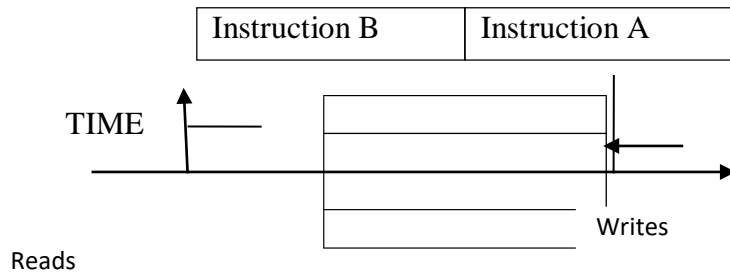
Handling Data hazard

- Operand forwarding
- Handling data hazards by introducing NOP (software method)

- Step D₂ must be delayed to clock cycle 5, and is shown as step D_{2A} in the figure. Instruction I₃ is fetched in cycle 3, but its decoding must be delayed because step D₃ cannot precede D₂.
- Hence, pipelined execution is stalled for two cycles.

Operand Forwarding:

- Consider instruction A and B as illustrated in the figure, B tries to read a register before A has written it and gets the old value.

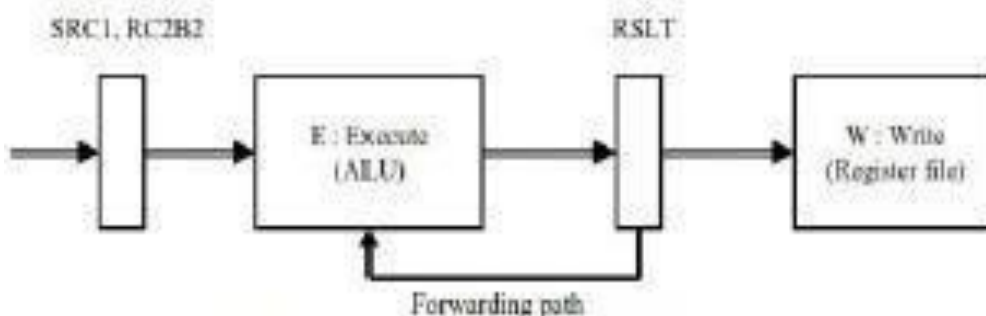
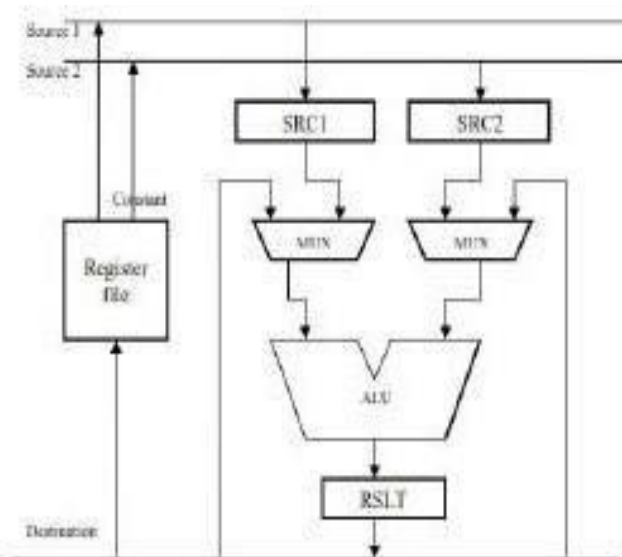


- This is quite common and called **read after write data hazard**. This situation is solved with a simple hardware technique called **operand forwarding**.

Example:

Add \$ s0, \$ t0, \$ t1
 Sub \$ t2, \$ s0, \$ t3

(A) Forwarding Datapath



(b) Position of the source and result registers in the processor pipeline.

Operand forwarding in a pipeline processor

Figure shows a part of the processor data path involving the ALU and the register file. This arrangement is similar to the three bus structure except that registers SRC1, SRC2 and RSLT have been added.

- These registers constitute inter stage buffers needed for pipelined operation. The two multiplexers connected at the inputs to the ALU allow the data on the destination bus to be selected instead of the contents of either the SRC1 or SRC2 register.
- After decoding instruction I_2 and detecting the data depending, a decision is made to use data forwarding. Register R_2 is read and loaded in register. SRC1 in clock cycle 3.
- In the next cycle, the product produced by instruction I_1 , is available in register RSLT and because of the forwarding connection it can be used in steep E_2 . Hence execution of I_2 proceeds without interruption.

Handling Data Hazard in Software:

- Another way to avoid data dependencies is to use software.
- In the software approach compiler can introduce two cycle delay needed between instruction I_1 and I_2 in figure by NOP(no operation) instruction as follows:

```

I1:      Mul    R2,R3,R4
        NOP
        NOP
I2:      Add    R5,R4,R6

```

- In the responsibility for detecting such dependencies is left entirely to the software the compiler must insert NOP instruction to obtain a correct result. This possibility illustrates the close link between the compiler and the hardware.

Side Effects:

- When a location other one explicitly named in an instruction as a destination operand is affected, the instruction is said to have a side effect.
- An instruction that uses an auto increment or auto decrement addressing mode is an example.
- In addition to storing a new data in its destination location, the instruction changes the contents of a source register used to access one of its operands.
- For example, a stack instructions, such as push and pop, produce similar side effects because they implicitly use the auto increment and auto decrement addressing modes
 - Another possible side effect involves the condition code flags, which are used by instructions such as conditional branches and add with carry.

Add R_1, R_3

Add with carry R_2, R_4 .

- An implicit dependency exists between these two instructions through the carry flag. This flag is set by the first instruction and used in the second instruction, which performs the operation.

$R_4 \leftarrow [R_2] + [R_4] + \text{carry}.$

- Instructions that have side effects give rise to multiple data dependencies, which lead to a substantial increase in the complexity of the hardware or software needed to resolve them. For this reason, instructions designed for execution on pipelined hardware should have few side effects.
- Ideally, only the content of the destination location, either a register or a memory location, should be affected by any given instruction. Side effects, such as setting the condition code flags or updating the contents of an address pointer, should be kept to a minimum.

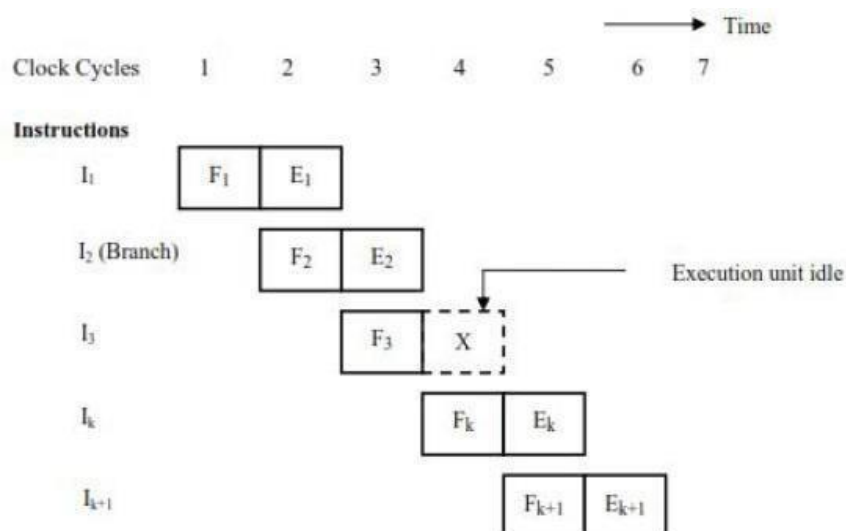
10. Explain How to handle Instruction Hazards or control hazards. (Nov/Dec2014)(Apr/May2015)Or Describe the techniques for handling control hazards in pipelining. (May/June2013) Or Explain the hazards caused by unconditional branching statements. (7 Marks) Apr / May 2017

Over View:

- Unconditional Branch
 - Conditional Branch
 - Branch Prediction
- This type of hazard arises from pipeline of branch and other instructions that change the contents of PC. (i.e) Trying to make a decision based on the results of instruction while others are executing.

Unconditional Branch:

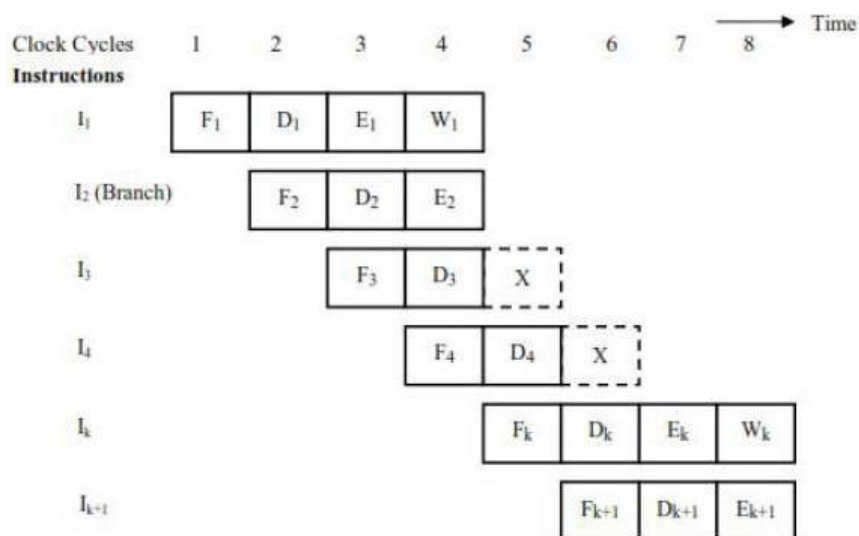
The below figure shows a sequence of instructions being executed in a two-stage pipeline instruction I_1 to I_3 are stored at consecutive memory address and instruction I_2 is a branch instruction.



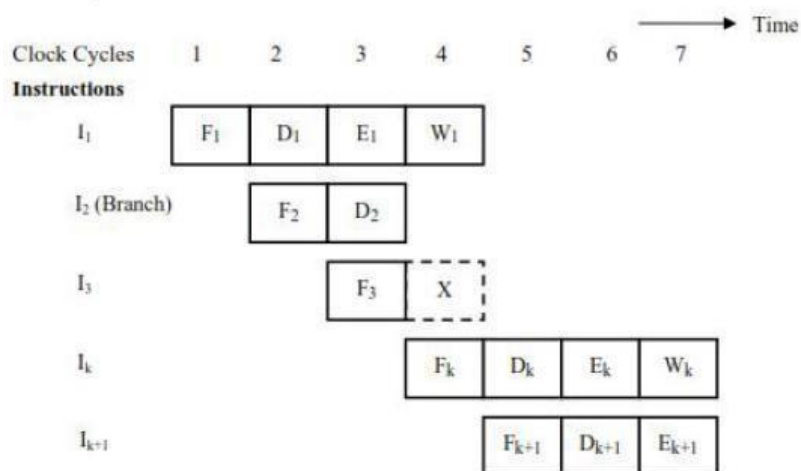
An idle cycle caused by a branch instruction

- If branch is taken as shown in figure, then the PC value is not known till the end of I_2 . Next three instructions are fetched even though they are not required. Hence they have to be flushed after branch is taken and new set of instructions have to be fetched from the branch address.

- In figure, clock cycle 3, the fetch operation for instruction I_3 is in progress at the same time the branch instruction is being decoded. In clock cycle 4, the processor must discard I_3 , which has been incorrectly fetched, and fetch instruction I_k . Thus the pipeline is stalled for one clock cycle.
- The time lost as a result of branch instruction is often referred to as the **branch penalty**.
- The branch penalties can be reduced by proper scheduling through compiler technique. The basic idea behind these techniques is to fill the 'delay slot' with some useful instruction which in most cases will be executed.
- For longer pipeline, the branch penalty may be higher. Reducing the branch penalty requires the branch address to be computed earlier in the pipeline.
- The instruction fetch unit has dedicated hardware to identify a branch instruction and compute branch target address as quickly as possible after an instruction is fetched.
- With these additional hardware both these tasks can be performed in step D_2 , leading to the sequence of events shown in figure. In this case the branch penalty is only one clock cycle.



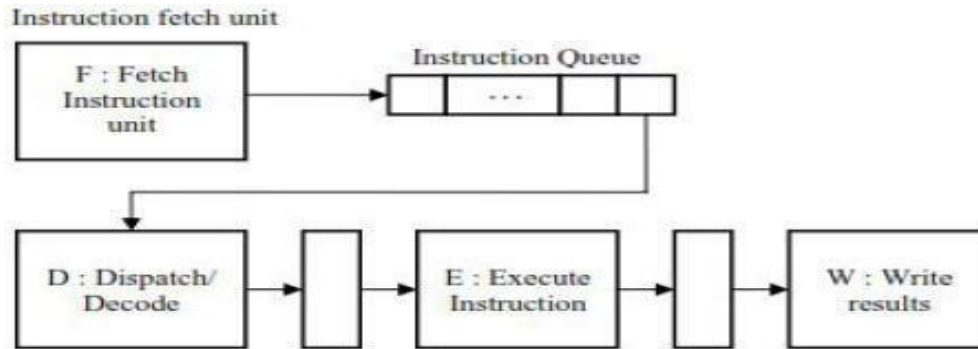
Branch address computed in execute stage



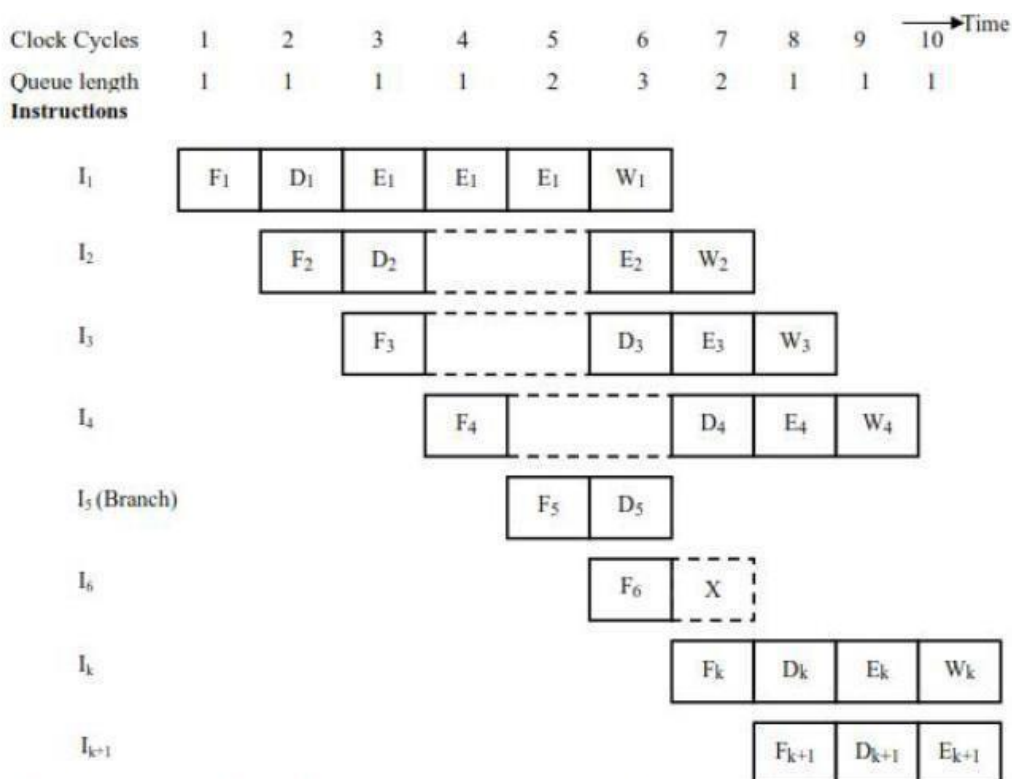
Branch address computed in decode stage

Instruction Queue and Pre fetching:

- The **Fetch unit** may contain instruction queue to store the instruction before they are needed to avoid interruption.
- Another unit called **dispatch unit** takes instruction from the front of the queue and sends them to the section unit. The dispatch unit also performs the decoding function.



Hardware organization of instruction queue



Branch timing in the presence of an instruction queue. Branch target address in computed D stage

- The fetch unit must have sufficient decoding and processing capability to recognize and execute branch instruction.
- The fetch unit always keeps the instruction queue filled at all times.
- Fetch unit continues to fetch instructions and add them to the queue.
- Similarly if there is a delay in fetching instructions, the dispatch unit continues to issue instruction from the instruction queue.
- Every fetch operation adds one instruction to the queue and each dispatch operation reduces queue length by one. Hence queue length remains same for first four clock cycle.

- Instruction I_5 is a branch instruction Its target instruction, I_k , is fetched in cycle 7, and instruction I_6 is discarded. The branch instruction would normally cause a stall in cycle 7 as a result of discarding instruction I_6 . Instead, instruction I_4 is dispatched from queue to the decoding stage. After discarding I_6 , the queue length drops to 1 in cycle 8. The queue length will be at this value until another stall is encountered.
- The sequence of instruction completions instruction I_1, I_2, I_3, I_4 and I_k complete execution in successive clock cycle. Hence the branch instruction does not increase the overall execution time.
- This is because the instruction fetch unit has executed branch instruction concurrently with the execution of other instruction. This technique is referred to as **branch folding**.
- Branch folding occurs only if at the time a branch instruction encountered, at least one instruction is available in the queue other than the branch instruction.

11. Explain about Conditional Branches: (Apr/May2014)

Conditional Branches:

- The conditional branching is a major factor that affects the performance of instruction pipelining. When a conditional branch is executed it may or may not change the PC.
- If a branch changes the PC to its target address, it is a taken branch, if it falls through, it is not taken. The decision to branch cannot be taken until the execution of that instruction has been completed.

Delayed Branch:

- The location following the branch instruction is called branch delay slot. There may be more than one branch delay slot depending on the time it takes to execute the branch instruction.
- The instruction in the delay slot is always fetched at least partially executed before the branch decision is made and the branch target address is completed.
- A technique called **delayed branching** can minimize the penalty caused by conditional branch instruction.
- The instruction in the delay slot is always fetched. Therefore, arrange the instructions which are fully executed, whether or not the branch is taken. Place the useful instructions in the delay slot. If no useful instructions available; fill the slot with NOP instructions.

EXAMPLE:

```

-----
LOOP      shift-Left      R1
          Decrement      R2
          Branch = 0      LOOP
NEXT      Add             R1, R2

```

(a). Original program loop

```

-----
LOOP      Decrement      R2

```

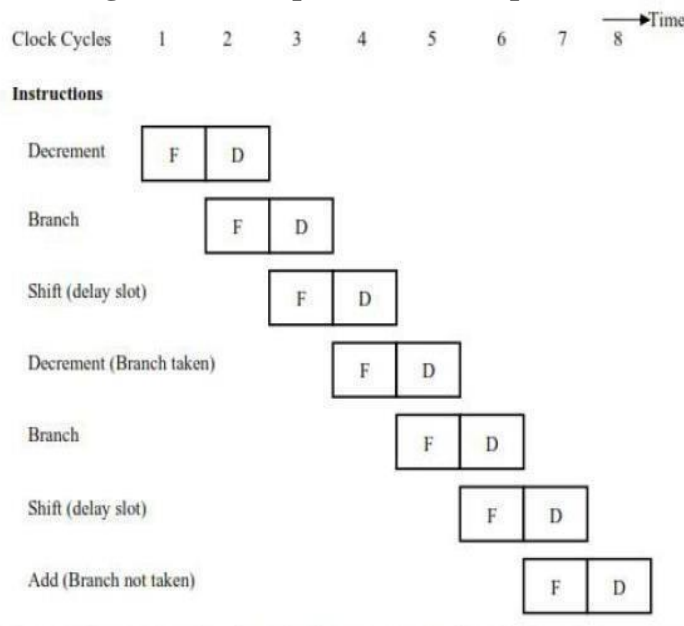

	Branch = 0	LOOP
	shift-Left	R ₁
NEXT	Add	R ₁ , R ₂

(b). Reordering instructions

Reordering of instructions for a delayed branch

- Register R₂ is used as counter to determine the number of times contents of R₁ are shifted left. For a processor with one delay slot, the instructions can be recorded a above. For a processor with one delay slot, the instructions can be reordered as shown in above figure(b).
- The shift instruction is fetched while branch instruction is being executed.
- After evaluating the branch condition, the processor fetches the instruction at LOOP or at NEXT, depending on whether the branch condition is true or false respectively. In either case, it completes the execution of the shift instructions.

The sequence of events during the last two passes in the loop is illustrated in figure.



Execution timing showing the delay slot being filled during two passes through the loop

- Pipelined operation is not interrupted at any time, and there are no idle cycles. Branching takes place one instruction later than where branch instruction appears in the sequence, hence named "delayed branch".

12. Explain about Branch prediction Algorithm. Nov / Dec 2016

Branch prediction:

Over view:

- Speculative execution
- Static prediction
- Dynamic Branch Prediction

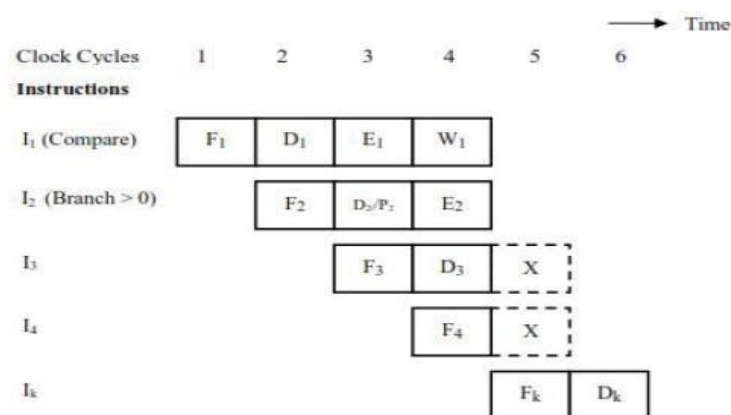
Branch prediction

- Prediction techniques can be used to check whether a branch will be valid or not valid. The simplest form of branch prediction is to assume that the branch will not take place and to continue to fetch instructions in sequential address order. Until the branch condition is evaluated, instruction execution along the predicted path must be done on a speculative basis.

Speculative execution means that instructions are executed before the processor is certain that they are in the correct execution sequence.

The below figure illustrate the incorrectly predicted branch.

- Figure shows a compare instruction followed by Branch > 0 instruction. In cycle 3 the branch prediction takes; the fetch unit predicts that branch will not be taken and it continues to fetch instruction I₄ as I₃ enters the Decode Stage.
- The result of compare operation is available at the ends of cycle 3. The branch condition is evaluated in cycle 4. At this point, the instruction fetch unit realizes that the prediction was incorrect and the two instructions in the execution pipe are purged.
- A new instruction I_k is fetched from the branch target address in clock cycle 5. We will examine prediction schemes static and dynamic prediction.



Timing when branch decision has been incorrectly predicted as not taken

Static prediction

- Static prediction is usually carried out by the compiler and it is static because the prediction is already known even before the program is executed.

Dynamic Branch Prediction: (May/June 2013)

- Dynamic prediction in which the prediction **may change depending on execution** history.

Algorithm:

- If the branch taken recently, the next time if the same branch is executed, it is likely that the branch is take.

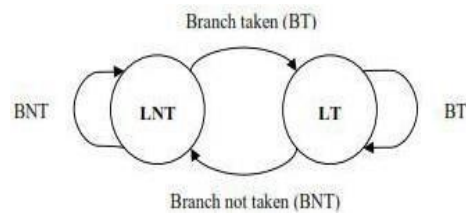
State 1:LT: Branch is likely to be take.

State 2:LNT:Branch is likely not to be take.

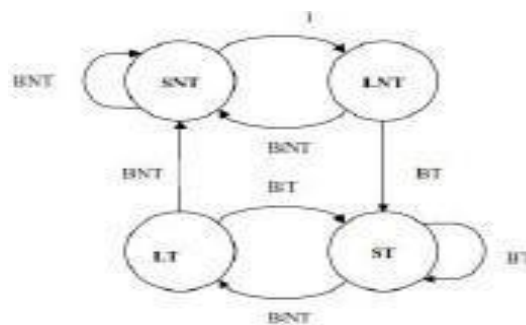
The algorithm is stated in state LNT when the branch is executed.

- If the branch is taken, the machine moves to LT. Otherwise it remains in state LNT.

- The branch is predicted as taken if the corresponding state machine is in state LT, otherwise it is predicted as not take.
- The branch is predicted as taken if the corresponding state machine is in state LT, otherwise it is predicted as not take.



A 2-State machine representation of branch-prediction



A 4-State machine representation of branch-prediction

Algorithm:

- An algorithm that uses 4 states, thus requiring two bits of history information for each branch instruction is shown in figure. The four states are:

ST : Strongly likely to be taken

LT : Likely to be taken.

LNT : Likely not to be taken

SNT : Strongly likely not to be taken.

STEP 1: Assume that the state of algorithm is initially set to LNT.

STEP 2: If the branch is actually taken change to ST, otherwise it is changed to SNT

STEP 3: When a branch instruction is encountered, the branch will be taken if the state is either LT or ST and it begins to fetch instructions at the branch target address. Otherwise, it continues to fetch instructions in sequential address order.

- When in state SNR, the instruction fetch unit predicts that the branch will not be taken.
- If the branch is actually taken, that is if the prediction is incorrect, the state changes to LNT.

The state information used in dynamic branch prediction algorithm requires 2 bits for 4 states and may be kept by the processes in a variety of ways,

- Use look-up table, which is accessed using low-order part of the branch of instruction address.
- Store as tag bits associated with branch instruction in the instruction cache.

PART-A

1. What is MIPS and write its instruction set?

MIPS is a reduced instruction set **computer** (RISC) instruction set **architecture** (ISA) developed by MIPS Technologies (formerly MIPS Computer Systems). The early MIPS architectures were 32-bit, with 64-bit versions added later.

MIPS instruction set:(Micro Instruction per Second)

- **The memory**-reference instructions load word (lw) and store word (sw)
- **The arithmetic**-logical instructions add, sub, AND, OR, and slt
- **The instructions**-branch equal (beq) and jump (j), which we add last.

2. What are R-type instructions? (Apr/May 2015)Nov/Dec 2020

MIPS fields are given names to make them easier to discuss:

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Here is the meaning of each name of the fields in MIPS instructions:

- **op**:Basic operation of the instruction, traditionally called the **opcode**.
- **rs**:The first register source operand.
- **rt**:The second register source operand.
- **rd**:The register destination operand. It gets the result of the operation.
- **shamt**:Shift amount.
- **funct**: Function. This field, often called the *function code*, selects the specific variant of the operation in the op field.

3. Define Branch target address.

- The address specified in a branch, which becomes the new program counter (PC) if the branch is taken. In the MIPS architecture the branch target is given by the sum of the offset field of the instruction and the address of the instruction following the branch.

4. Define the terms Data path element, CPU Data path and Data path cycle? Nov / Dec 2016, Apr. / May 2018 Nov/Dec 2020.

- A unit used to operate on or hold data within a processor. In the MIPS implementation, the data path elements include the instruction and data memories, the register file, the ALU and adders.
- The path that data follows within the CPU, along buses from registers to ALU and back is called the **CPU Datapath**.
- Everything a computer does, whether playing an MPEG file, or a video game, is, in the end, essentially a sequence of very primitive operations whereby data is moved from registers to the ALU, operations are performed on that data, and then the result is returned to the registers. A single round of Registers -> ALU -> Registers is called a **CPU Datapath Cycle**.

5. When will the instruction have die effect?

- Sometime an instruction changes the contents of a register other than the destination. An instruction that uses an auto increment or auto decrement addressing mode is an example.
- Add with Carry R2, R4
- This instruction will take the carry value present in the condition code register. So it refers the register which is not represented in the instruction

6. Define branch penalty.

- The time lost as a result of a branch instruction is often referred to as the branch penalty. This will cause the pipeline to stall. So we can reduce branch penalty by calculating the branch address in early stage.

7. What is the use of instruction queue in pipeline?

- Many processors can fetch the instruction before they are needed and put them in queue is called instruction queue. This instruction queue can store several instructions.

8. Define dispatch unit.

- It is mainly used in pipeline concept. It takes the instruction from the front of the instruction queue and sends them to the execute unit for execution.

9. What is meant by branch folding and what is the condition to implement it?

- The instruction fetch unit has executed the branch instruction concurrently with in the execution of other instructions is called branch folding.
- This occurs only if at the time of branch is encountered at least one instruction is available in the queue than the branch instruction.

10. What is meant by delay branch slot?

- A location following branch instruction is called as branch delay slot. There may be more than one branch delay slot, depending on the execution time.
- The instruction in the delay slot is always fetched and at least partially executes before the branch decision is made.

11. Define delayed branching.

- It is a technique by using it we can handle the delay branch slot instructions. We can place some useful instruction in the branch delay slot and execute these instructions when the processor is executing the branch instruction.
- If there is no useful instruction in the program we can simply place NOP instruction in delay slot. This technique will minimize the branch penalty.

12. Define branch prediction. Nov / Dec 2015

It is a technique used for reducing branch penalty associated with the condition branches. Assume that the branch will not take place and to continue the fetch instructions in sequential address order until the branch condition is evaluated.

13. What are the two types of branch prediction technique available? (May/June 2009)

The two types of branch prediction techniques are

- Static branch prediction
- Dynamic branch prediction

14. Define static and dynamic branch prediction.

- The branch prediction decision is always the same for every time a given instruction is executed. This is known as static branch prediction.
- Another approach in which the prediction may change depending on execution history is called dynamic branch prediction.

15. List the two states in the dynamic branch prediction.

- LT : Branch is likely to be taken.
- LNT : Branch is likely not to be taken.

16. List out the four stages in the branch prediction algorithm.

- ST : Strongly likely to be taken
- LT : Likely to be taken
- LNT : Likely not to be taken
- SNT : Strongly not to be taken

17. Define Register renaming. (Nov/Dec 2009)

- When temporary register holds the contents of the permanent register, the name of permanent register is given to that temporary register is called as **register renaming**.
- **For example**, if I2 uses R4 as a destination register, then the temporary register used in step TW2 is also referred as R4 during cycles 6 and 7 that temporary register used only for instructions that follow I2 in program order.
- For example, if I1 needs to read R4 in cycle 6 or 7, it has to access R4 though it contains unmodified data by I2.

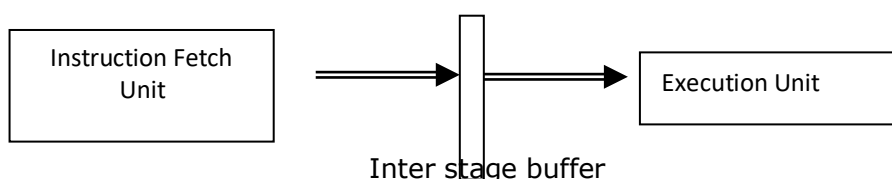
18. What is pipelining and what are the advantages of pipelining? (Apr/May 2010) Nov / Dec 2013

- Pipelining is process of handling the instruction concurrently.
- The pipelining processor executes a program by one after another.

Advantages: May / June 2016

- Pipelining improves the overall throughput of an instruction set processor.
- It is applied to design of complex data path units such as multiplexers and floating points adders.

19. Draw the hardware organization of two-stage pipeline.



20. Name the four stages of pipelining. (Or)What are the steps in pipelining processor?Nov/Dec 2020.

Fetch	:	Read the instruction from the memory.
Decode	:	Decode the instruction and fetch the source operands.
Execute	:	Perform the operation specified by the instruction
Write	:	Store the result in the destination location.

21. Write short notes on instruction pipelining.

- The various cycles involved in the instruction cycle.
- These fetch, decode and execute cycles for several instructions are performed simultaneously to reduce overall processing time.
- This process is referred as **instruction pipelining**.

22. What is the role of cache in pipelining? (Or) What is the need to use the cache memory in pipelining concept?(Nov/Dec 2011)

- Each stage in a pipeline is expected to complete its operation in one clock cycle. But the accessing time of the main memory is high.
- So it will take more than one clock cycle to complete its operation. So we are using cache memory for pipelining concept.
- The accessing speed of the cache memory is very high.

23. What is meant by bubbles in pipeline? Or what is meant by pipeline bubble? Nov / Dec 2016

- Any condition that causes the pipeline to be idle is known as pipeline stall. This is also known as bubble in the pipeline. Once the bubble is created as a result of a delay, a bubble moves down stream until it reaches the last unit.

24. What are the major characteristics of pipeline?

- Pipelining cannot be implemented on a single task, as it works of splitting multiple tasks into a number of subtasks and operating on them simultaneously.
- The speedup or efficiency is achieved by using a pipeline depends on the number of pipe stages and the number of available tasks that can be subdivided.

25. Give the features of the addressing mode suitable for pipelining. (Apr/May 2014)

- They access operand from memory in only one cycle.
- Only load and store instruction are provided to access memory.
- The addressing modes used do not have side effects.(When a location other than one explicitly named in an instruction as the destination operand is affected, the instruction is said to have a side effect).
- Three basic addressing modes used do not have these features are register, register indirect and index. The first two require bus address computation. In the index mode, the address can be computed in one cycle, whether the index value is given in the instruction or in registration.

26. What are the disadvantages of increasing the number of stages in pipelined processing?(Apr/May 2011) (Or) What would be the effect,if we increase the number of pipelining stages? (Nov/Dec 2011)

Speedup:

Speedup is defined by

$$S(m)=T(1)/T(m)$$

- **Where T (m)** is the execution time for some target workload on an m-stage pipeline and **T(1)** is the execution time for same target workload on a non-pipelined Processor.

27. What is the ideal CPI of a pipelined processor?

The ideal CPI on a pipelined processor is almost always 1. Hence, we can compute the pipelined CPI:

CPI pipelined = Ideal CPI + Pipeline stall clock cycles per instruction = 1 + Pipeline stall clock cycles per instruction

28. How can memory access be made faster in a pipelined operation? Which hazards can be reduced by faster memory access? (Apr/May 2010)

The goal in controlling a pipelined CPU is maximize its performance with respect to the target workloads.

Performance measures:

The various performance measures of pipelining are,

- Throughput
- CPI
- Speedup
- Dependencies
- Hazards

The following Hazards can be reduced by faster memory access:

- Structural hazards
- Data or Data dependent hazards
- Instruction or control hazards

29. Write down the expression for speedup factor in a pipelined architecture. May 2013

- The speedup for pipeline computer is

$$S=(K+n-1)t_p$$

Where,

k-number of segments in a pipeline.

n-number of instruction to be executed.

t_p - cycle time.

30. Define Hazard and State different types of hazards that occur in pipeline. Nov / Dec 2015, Apr / May 2017, May 2019 Nov/Dec 2020.

In the domain of central processing unit (CPU) design, **hazards** are problems with the instruction pipeline in CPU microarchitectures when the next instruction cannot execute in the following clock cycle, and can potentially lead to incorrect computation results.

The various pipeline hazards are:

- Structural hazards
- Data or Data dependent hazards
- Instruction or control hazards

31. What is structural hazard?(Nov/Dec 2008) (Apr /May 2014)

- When two instructions require the use of a given hardware resource at the same time this hazard will occur. The most common case of this hazard is memory access.

32. What is data hazard in pipelining? (Nov/Dec 2007, 2008)

- A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in pipeline. As a result some operation has be delayed and the pipeline stalls.
- Arise when an instruction depends on the results of a previous instruction in a way that is exposed by overlapping of instruction in pipeline

33. What are instruction hazards (or) control hazards?

- They arise while pipelining branch and other instructions that change the contents of program counter.
- The simplest way to handle these hazards is to stall the pipeline stalling of the pipeline allows few instructions to proceed on completion while stopping the execution of those which results in hazards.

34. How can we eliminate the delay in data hazard?

- In pipelining the data can be executed after the completion of the fetch operation. The data are available at the output of the ALU once the execute stage completes.
- Hence the delay can be reduced if we arrange for the result of fetch instruction to be forwarded directly for use in next step. This is known as operand forwarding.

35. How can we eliminate data hazard using software?

- The data dependencies can be handled with the software. The compiler can be used for this purpose. The compiler can introduce the two cycle delays needed between instruction I1 and I2 by inserting NOP (no operation)

I₁: MUL R2, R3, R4

NOP

NOP

I₂: ADD R5, R4, R6

36. List the techniques used for overcoming hazard.

- Data forwarding
- Adding sufficient hardware
- Stalling instructions
- Document to find instruction in wrong order.

37. What are the techniques used to present control hazard?

- Scheduling instruction in delay slots
- Loop unrolling
- Conditional execution
- Speculation (by both compiler and CPU).

38. List the types of data hazards.

- RAW (Read After Write)
- WAW (Write After Write)
- WAR (Write After Read)
- RAR (Read After Read)

39. Define stall.

Idle periods are called stalls. They are also often referred to as bubbles in the pipeline.

40. Give 2 examples for instruction hazard.

- Cache miss
- Hazard in pipeline.

41. $A = 5$ $A < -3 + A$ $A < -4 + A$ What hazard does the above two instructions create when executed concurrently? (Apr/May 2011)

If these operations are performed in the order given, the result is 32. But, if they were performed concurrently, the value is 5. So output is wrong.

42. What is meant by speculative execution? (Apr/May 2012) Or what is the need for speculation? (Nov/Dec 2014), May 2019

- A technique allows a superscalar processor to keep its functional units as busy as possible by executing instructions before it is known that they will be needed.
- The Intel P6 uses speculative execution.

43. What is meant by hazard in pipelining? Define data and control hazards. (May/June 2013) (Apr/May 2012)

- The idle periods in any of the pipeline stage due to the various dependency relationships among instructions are said to be stalls.
- A **data hazard** is any condition in which either the source or the destination operands of an instruction are not available at the time expected in pipeline.
- As a result some operation has be delayed and the pipeline stalls arise when an instruction depends on the results of a previous instruction in a way that is exposed by overlapping of instruction in pipeline.

Types

1. RAW 2. WAW 3. WAR

- **Control hazards** arise while pipelining branch and other instructions that change the contents of program counter. The simplest way to handle these hazards is to stall the pipeline stalling of the pipeline allows few instructions to proceed to completion while stopping the execution of those which results in hazards

44. Why is branch prediction algorithm needed? Distinguish between static and dynamic branch prediction. (May/June 2009) Or Differentiate between the static and dynamic techniques. (May/June 2013)

Branch Prediction has become essential to getting good performance from scalar instruction streams.

- Underlying algorithm has regularities.

- Data that is being operated on has regularities.
- Instruction sequence has redundancies that are artifacts of way that humans/compiler think about problems.

S.NO.	STATIC BRANCH PREDICTION	DYNAMIC BRANCH PREDICTION
1.	Branch can be predicted based on branch codes type statistically.	It used recent branch history during program execution, information is stored in buffer called branch target buffer(BTB).
2.	It may not produce accurate result every time.	Processing of conditional branches with zero delay.

45. What is Branch Target Buffer?

Branch Target Buffer (BTB): A hardware mechanism that aims at reducing the stall cycles resulting from correctly predicted taken branches to zero cycles.

46. Define program counter (PC).

- The register containing the address of the instruction in the program being executed.

47. What are Sign-extend?

- To increase the size of a data item by replicating the high-order sign bit of the original data item in the high order bits of the larger, destination data item.

48. Define Register file.

- A state element that consists of a set of registers that can be read and written by supplying a register number to be accessed.

49. What is a Don't-care term?

- An element of a logical function in which then output does not depend on the values of all the inputs.

50. Define Forwarding.

A method of resolving a data hazard by retrieving the missing data element from internal buffers rather than waiting for it to arrive from programmer visible registers or memory. Also called **bypassing**.

51. What is a branch prediction buffer? (Apr/May 2015)

A small memory that is indexed by the lower portion of the address of the branch instruction and that contains one or more bits indicating whether the branch was recently taken or not. It is also called branch **history table**.

52. What is an Exception? Nov / Dec 2014, May / June 2016, Apr. / May 2018, Nov. / Dec. 2018

Exceptions and interrupts are unexpected events that disrupt the normal flow of instruction execution. An **exception** is an unexpected event from within the processor. An interrupt is an unexpected event from outside the processor. We have to implement **exception** and interrupt handling in our multi cycle CPU design.

53. Give one example for MIPS exception. Apr. / May 2018, Nov. / Dec. 2018

Exceptions in MIPS

stage	Problem exceptions occurring
IF	Page fault on IF, misaligned memory access, memory protection violation
ID	Undefined or illegal opcode
EX	Arithmetic exception
MEM	Page fault on data fetch, misaligned memory access, memory protection violation
WB	None

54. What is precise and imprecise exception? (Apr/May 2009)(Nov/Dec 2019)

- A **precise exception** is one in which all instruction prior to the faulting instruction are complete and instruction following the instruction, including the faulting instruction do not change the state of the machine. (Or)
- If the execution occurs during an instruction, all subsequent instructions that may have been executed are discarded. This is called **precise exception**.
- If one instruction causes an exception and succeeding instructions are permitted to complete execution, then the processor is said to have **imprecise exception**.

55. Define edge triggered clocking. May 2019(Nov/Dec 2019)

A falling **edge** is the high to low transition. It is also known as the negative **edge**. When a circuit is falling **edge-triggered**, it becomes active when the **clock** signal goes from high to low, and ignores the low-to-high transition. A leading **edge** is an event that is **triggered** on the front **edge** of a pulse.

56. What is Instruction Level Parallelism? (Dec 2012, Dec 2013, May 2015, May 2016)

- The technique which is used to overlap the execution of instructions and improve performance is called ILP.

57. What are the approaches to exploit ILP? (Dec 2012, Dec 2015)

The two separable approaches to exploit ILP are,

- Dynamic or Hardware Intensive Approach
- Static or Compiler Intensive Approach.

58. What is Loop Level Parallelism?

- Loop level parallelism is a way to increase the amount of parallelism available among instructions is to exploit parallelism among iterations of loop.

59. Give the methods to enhance performance of ILP.

To obtain substantial performance enhancements, the ILP across multiple basic blocks are exploited using

- Loop Level Parallelism
- Vector Instructions

60. Define Dynamic Scheduling. (May 2013) (Or) Explain the idea behind dynamic scheduling. (Nov/Dec 2016)

- Dynamic scheduling is a technique in which the hardware rearranges the instruction execution to reduce the stalls while maintaining data flow and exception behavior.

61. List the drawbacks of Dynamic Scheduling.

- The complexity of the tomasulo scheme.
- Each reservation station must contain an associative buffer.
- The performance can be limited by the single CDB.

62. List the advantages of Dynamic Scheduling. (May 2012)

- It handles dependences that are unknown at compile time.
- It simplifies the compiler.
- It allows code compiled for one pipeline to run efficiently on a different pipeline
- Uses speculation techniques to improve the performance.

63. Differentiate Static and Dynamic Scheduling.

Static Scheduling	Dynamic Scheduling
<ul style="list-style-type: none"> • The data hazard that prevents a new instruction issue in the next cycle was resolved using a technique called data forwarding • And also by compiler scheduling that separated the dependent instruction this is called as static scheduling. 	<ul style="list-style-type: none"> • The CPU rearranges the instructions to reduce stalls while preserving dependences. • It uses hardware based mechanism to rearrange instruction execution order to reduce stalls at runtime. • It enables handling cases where dependences are unknown at compile time.

64. Define Dynamic Scheduling using Tomasulo's algorithm.

- **Tomasulo's algorithm** is a computer architecture hardware **algorithm** for **dynamic scheduling** of instructions that allows out-of-order execution and enables more efficient use of multiple execution units.

65. What is Branch Prediction?

- In computer architecture, a **branch predictor** is a digital circuit that tries to guess which way a **branch** (e.g. an if-then-else structure) will go before this is known definitively.
- The purpose of the branch predictor is to improve the flow in the instruction pipeline.

66. What are the types of branch prediction?

There are two types of branch prediction. They are,

- Dynamic Branch Prediction & Static Branch Prediction

67. What is meant by dynamic branch prediction? [May 2019]

- **Branch prediction** is used to overcome the fetch limitation imposed by control hazards in order to expose instruction-level parallelism (ILP).

- It is the key ingredient to pipelined and superscalar architectures that mask instruction execution latencies by exploiting (ILP).

68. What is Branch Prediction Buffer? (May 2014)

- Branch prediction buffer is a small memory indexed by the lower portion of the address of the branch instruction.
- The memory contains a bit that says whether the branch was recently taken or not.

69. What are the things present in Dynamic Branch Prediction?

It uses two things they are,

- Branch Prediction Buffer & Branch History Table

70. Define Correlating Branch Prediction.

- Branch prediction that uses the behavior of other branches to make a prediction is called correlating branch prediction.

71. List the five levels of branch prediction. (May 2013)

- Perfect
- Tournament Based Branch Predictor
- Standard Two Bit Branch Predictor with 512 - 2 Bit Entries
- Profile Based
- None

72. What is Reservation Station?

- In Tomasulo's scheme, register renaming is provided by reservation station.
- The basic idea is that the reservation station fetches and buffers an operand as soon as it is available, eliminating the need to get the operand from a register.

73. What is ROB?

- ROB stands for Reorder Buffer.
- It supplies operands in the interval between completion of instruction execution and instruction commit. ROB is similar to the store buffer in Tomasulo's algorithm.

74. What are the four fields involved in ROB?

ROB contains four fields,

- Instruction Type
- Destination Field
- Value Field
- Ready Field

75. What is Imprecise Exception?

- An exception is imprecise if the processor state when an exception is raised does not look exactly as if the instructions were executed sequentially in strict program order.

76. What are the two possibilities of imprecise exceptions?

- If the pipeline has already completed instructions that are later in program order then that instruction will cause exception.

- If the pipeline has not yet completed instructions that are earlier in program order then that instructions will cause exception.

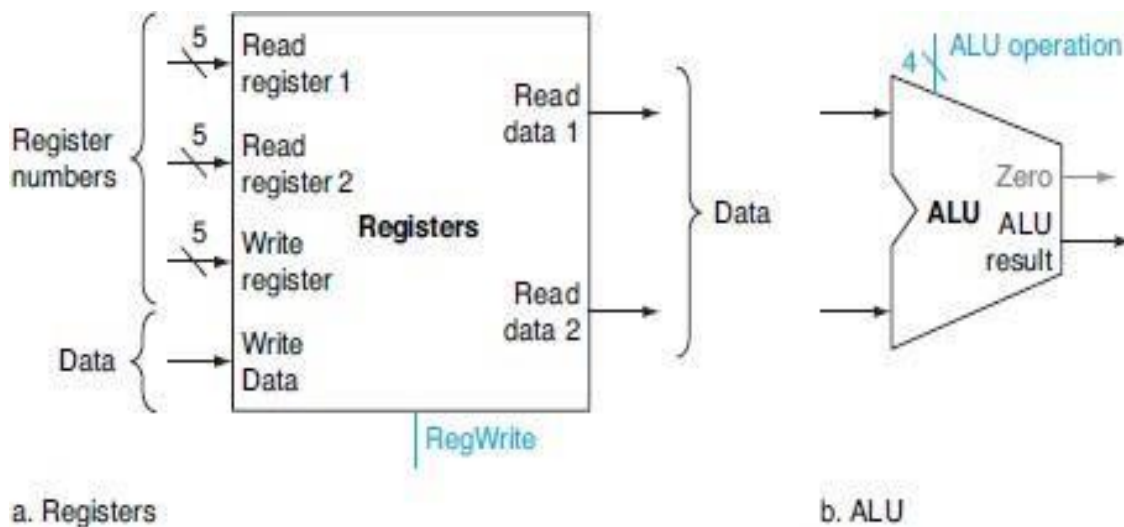
77. What is Register Renaming?

- Renaming of register operand is called register renaming.
- It can be either done statically by the compiler or dynamically by the hardware.

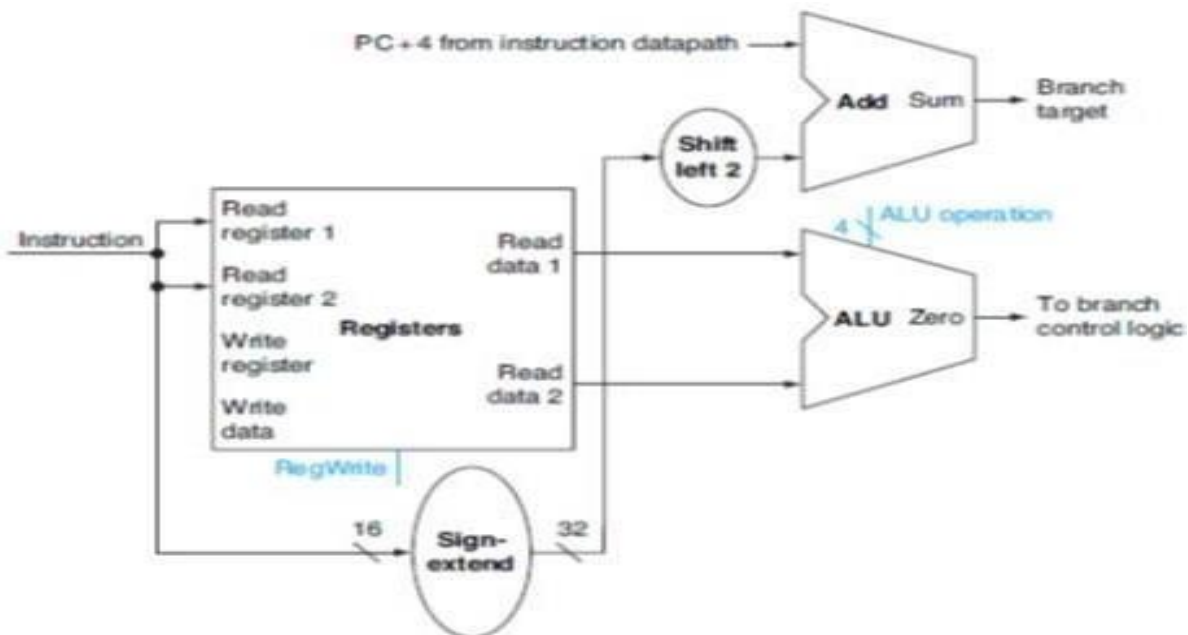
78. Difference between Static and Dynamic Branch Prediction? (May 2011)

Static Branch Prediction	Dynamic Branch Prediction
<ul style="list-style-type: none"> • Static branch prediction is usually carried out by the compiler. • It is static because the prediction is already known even before the program is executed. 	<ul style="list-style-type: none"> • It uses the run time behavior of branch to make more accurate prediction. • Information about the outcome of previous occurrences of a given branch is used to predict the current occurrences.

79. In a datapath diagram, what is the size of ALUop Control signal. Nov/Dec 2021



80. How PCSrc Signal generated in a datapath diagram? Nov/Dec 2021



UNIT V MEMORY AND I/O

Memory Concepts and Hierarchy – Memory Management – Cache Memories: Mapping and Replacement Techniques – Virtual Memory – DMA – I/O – Accessing I/O: Parallel and Serial Interface – Interrupt I/O – Interconnection Standards: USB, SATA

MEMORY CONCEPTS AND HIERARCHY

1. Explain about the memory concepts and hierarchy.

What is Memory?

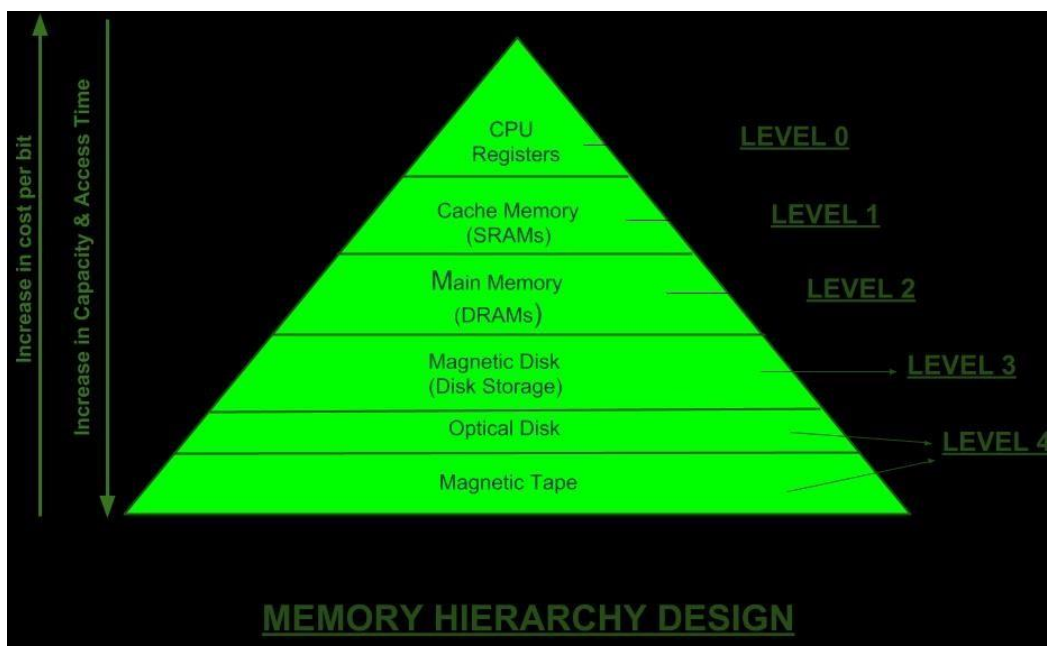
Computer memory is just like the human brain. It is used to store data/information and instructions. It is a data storage unit or a data storage device where data is to be processed and instructions required for processing are stored. It can store both the input and output can be stored here.

Characteristics of Main Memory:

- It is faster computer memory as compare to secondary memory.
- It is semiconductor memories.
- It is usually a volatile memory.
- It is the main memory of the computer.
- A computer system cannot run without primary memory.

Memory Hierarchy

In the Computer System Design, Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time. The Memory Hierarchy was developed based on a program behavior known as locality of references. The figure below clearly demonstrates the different levels of memory hierarchy :



This Memory Hierarchy Design is divided into 2 main types:

1. **External Memory or Secondary Memory** – Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.
2. **Internal Memory or Primary Memory** – Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

We can infer the following characteristics of Memory Hierarchy Design from above figure:

1. Capacity:

It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.

- 2. Access Time:** It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.

3. Performance:

Earlier when the computer system was designed without Memory Hierarchy design, the speed gap increases between the CPU registers and Main Memory due to large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.

- 4. Cost per bit:** As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

In general, memory is of three types:

- Primary memory
- Secondary memory
- Cache memory

Now we discuss each type of memory one by one in detail:

1. Primary Memory: It is also known as the main memory of the computer system. It is used to store data and programs or instructions during computer operations. It uses semiconductor technology and hence is commonly called semiconductor memory. Primary memory is of two types:

(i) RAM (Random Access Memory): It is a volatile memory. Volatile memory stores information based on the power supply. If the power supply fails/ interrupted/stopped, all the data & information on this memory will be lost. RAM is used for booting up or start the computer. It temporarily stores programs/ data which has to be executed by the processor. RAM is of two types:

- **S RAM (Static RAM):** It uses transistors and the circuits of this memory are capable of retaining their state as long as the power is applied. This memory consists of the number of flip flops with each flip flop storing 1 bit. It has less access time and hence, it is faster.

- **D RAM (Dynamic RAM):** It uses capacitors and transistors and stores the data as a charge on the capacitors. They contain thousands of memory cells. It needs refreshing of charge on capacitor after a few milliseconds. This memory is slower than S RAM.

(ii) ROM (Read Only Memory): It is a non-volatile memory. Non-volatile memory stores information even when there is a power supply failed/ interrupted/stopped. ROM is used to store information that is used to operate the system. As its name refers to read-only memory, we can only read the programs and data that is stored on it. It contains some electronic fuses that can be programmed for a piece of specific information. The information stored in the ROM in binary format. It is also known as permanent memory.

ROM is of four types:

- **MROM (Masked ROM):** Hard-wired devices with a pre-programmed collection of data or instructions were the first ROMs. Masked ROMs are a type of low-cost ROM that works in this way.
- **PROM (Programmable Read Only Memory):** This read-only memory is modifiable once by the user. The user purchases a blank PROM and uses a PROM program to put the required contents into the PROM. Its content can't be erased once written.
- **EPROM (Erasable Programmable Read Only Memory):** It is an extension to PROM where you can erase the content of ROM by exposing it to Ultraviolet rays for nearly 40 minutes.
- **EEPROM (Electrically Erasable Programmable Read Only Memory):** Here the written contents can be erased electrically. You can delete and re-programme EEPROM up to 10,000 times. Erasing and programming take very little time, i.e., nearly 4 -10 ms (milliseconds). Any area in an EEPROM can be wiped and programmed selectively.

2. Secondary Memory: It is also known as auxiliary memory and backup memory. It is a non-volatile memory and used to store a large amount of data or information. The data or information stored in secondary memory is permanent, and it is slower than primary memory. A CPU cannot access secondary memory directly. The data/information from the auxiliary memory is first transferred to the main memory, and then the CPU can access it.

Characteristics of Secondary Memory:

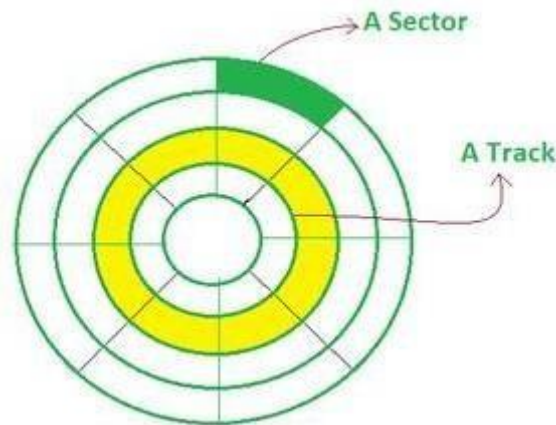
- It is a slow memory but reusable.
- It is a reliable and non-volatile memory.
- It is cheaper than primary memory.
- The storage capacity of secondary memory is large.
- A computer system can run without secondary memory.
- In secondary memory, data is stored permanently even when the power is off.

Types of secondary memory:

(i) Magnetic Tapes: Magnetic tape is a long, narrow strip of plastic film with a thin, magnetic coating on it that is used for magnetic recording. Bits are recorded on tape as magnetic patches called RECORDS that run along many tracks. Typically, 7 or 9 bits are recorded concurrently. Each track has one read/write

head, which allows data to be recorded and read as a sequence of characters. It can be stopped, started moving forward or backward, or rewind.

(ii) Magnetic Disks: A magnetic disc is a circular metal or a plastic plate and these plates are coated with magnetic material. The disc is used on both sides. Bits are stored in magnetized surfaces in locations called tracks that run in concentric rings. Sectors are typically used to break tracks into pieces.



Hard discs are discs that are permanently attached and cannot be removed by a single user.

(iii) Optical Disks: It's a laser-based storage medium that can be written to and read. It is reasonably priced and has a long lifespan. The optical disc can be taken out of the computer by occasional users.

Types of Optical Disks :

(a) CD – ROM:

- It's called Compact Disk. Only read from memory.
- Information is written to the disc by using a controlled laser beam to burn pits on the disc surface.
- It has a highly reflecting surface, which is usually aluminum.
- The diameter of the disc is 5.25 inches.
- 16000 tracks per inch is the track density.
- The capacity of a CD-ROM is 600 MB, with each sector storing 2048 bytes of data.
- The data transfer rate is about 4800KB/sec. & the new access time is around 80 milliseconds.

(b) WORM-(WRITE ONCE READ MANY):

- A user can only write data once.
- The information is written on the disc using a laser beam.
- It is possible to read the written data as many times as desired.
- They keep lasting records of information but access time is high.
- It is possible to rewrite updated or new data to another part of the disc.
- Data that has already been written cannot be changed.
- Usual size – 5.25 inch or 3.5 inch diameter.
- The usual capacity of 5.25 inch disk is 650 MB,5.2GB etc.

(c) DVDs:

- The term -DVD stands for -Digital Versatile/Video Disc, and there are two sorts of DVDs: (i) DVDR (writable) and (ii) DVDRW (Re-Writable)
- *DVD-ROMS (Digital Versatile Discs)*: These are read-only memory (ROM) discs that can be used in a variety of ways. When compared to CD-ROMs, they can store a lot more data. It has a thick polycarbonate plastic layer that serves as a foundation for the other layers. It's an optical memory that can read and write data.
- *DVD-R*: It is a writable optical disc that can be used just once. It's a DVD that can be recorded. It's a lot like WORM. DVD-ROMs have capacities ranging from 4.7 to 17 GB. The capacity of 3.5 inch disk is 1.3 GB.

3. Cache Memory: It is a type of high-speed semiconductor memory that can help the CPU run faster. Between the CPU and the main memory, it serves as a buffer. It is used to store the data and programs that the CPU uses the most frequently.

Advantages of cache memory:

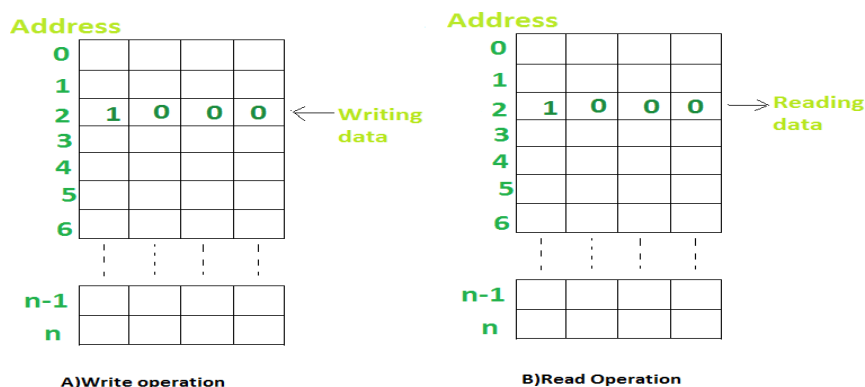
- It is faster than the main memory.
- When compared to the main memory, it takes less time to access it.
- It keeps the programs that can be run in a short amount of time.
- It stores data in temporary use.

Disadvantages of cache memory:

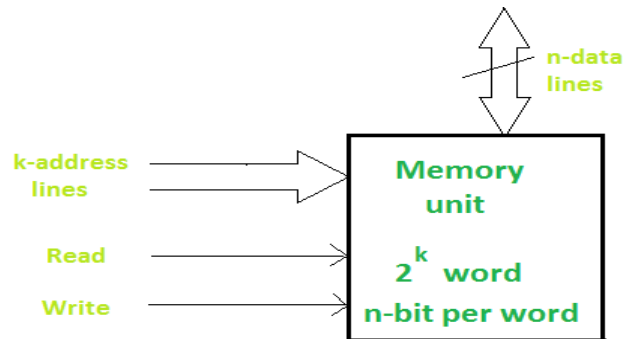
- Because of the semiconductors used, it is very expensive.
- The size of the cache (amount of data it can store) is usually small.

Memory unit:

- Memories are made up of registers.
- Each register in the memory is one storage location.
- The storage location is also called a memory location. Memory locations are identified using **Address**.
- The total number of bits a memory can store is its **capacity**.
- A storage element is called a **Cell**.
- Each register is made up of a storage element in which one bit of data is stored.
- The data in a memory are stored and retrieved by the process called **writing** and **reading** respectively.



- A **word** is a group of bits where a memory unit stores binary information.
- A word with a group of 8 bits is called a **byte**.
- A memory unit consists of data lines, address selection lines, and control lines that specify the direction of transfer. The block diagram of a memory unit is shown below:



- Data lines provide the information to be stored in memory.
- The control inputs specify the direct transfer.
- The k-address lines specify the word chosen.

When there are k address lines, 2^k memory words can be accessed.

Following are some important memory units:

- **Bit (Binary Units):** bit is a logical representation of the electric state. It can be 1 or 0.
- **Nibble:** it means the group of 4 bits.
- **Byte:** a byte is a group of 8 bits.
- **Word:** it is a fixed number of bits; it is different from computer to computer, but the same for each device. Compute store information in the form of words.

Following are conversations of units:

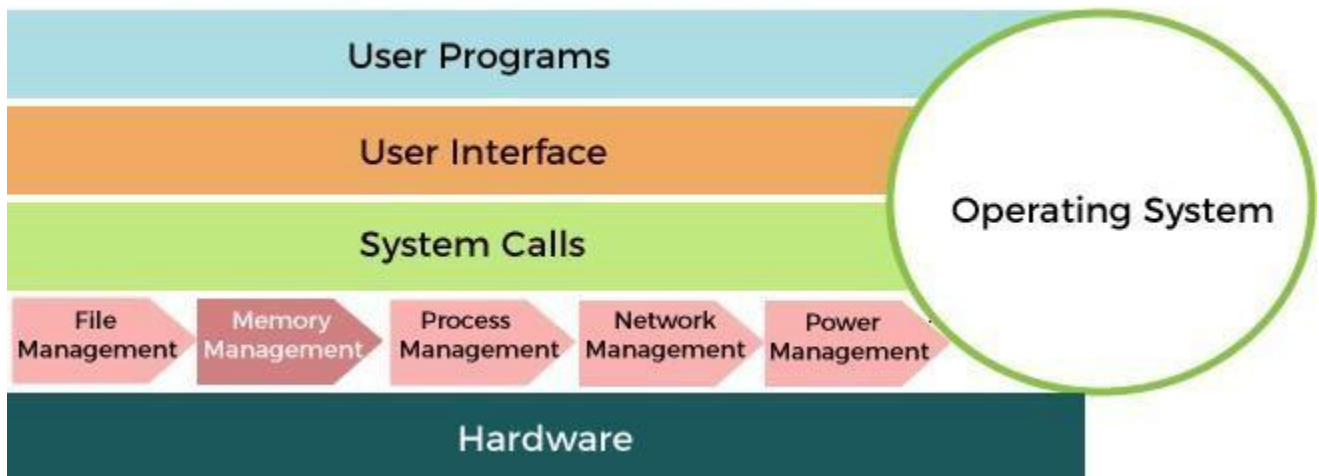
- **Kilobyte (kb):** 1kb = 1024 byte
- **Megabyte (mb):** 1mb = 1024 kb
- **Gigabyte (gb):** 1gb = 1024 mb
- **Terabyte (tb):** 1tb = 1024 gb
- **Petabyte (pb):** 1pb = 1024 tb

MEMORY MANAGEMENT

2. What do you mean by memory management?

Memory is the important part of the computer that is used to store the data. Its management is critical to the computer system because the amount of main memory available in a computer system is very limited. At any time, many processes are competing for it. Moreover, to increase performance, several processes are executed simultaneously. For this, we must keep several processes in the main memory, so it is even more important to manage them effectively.

Memory Management in OS



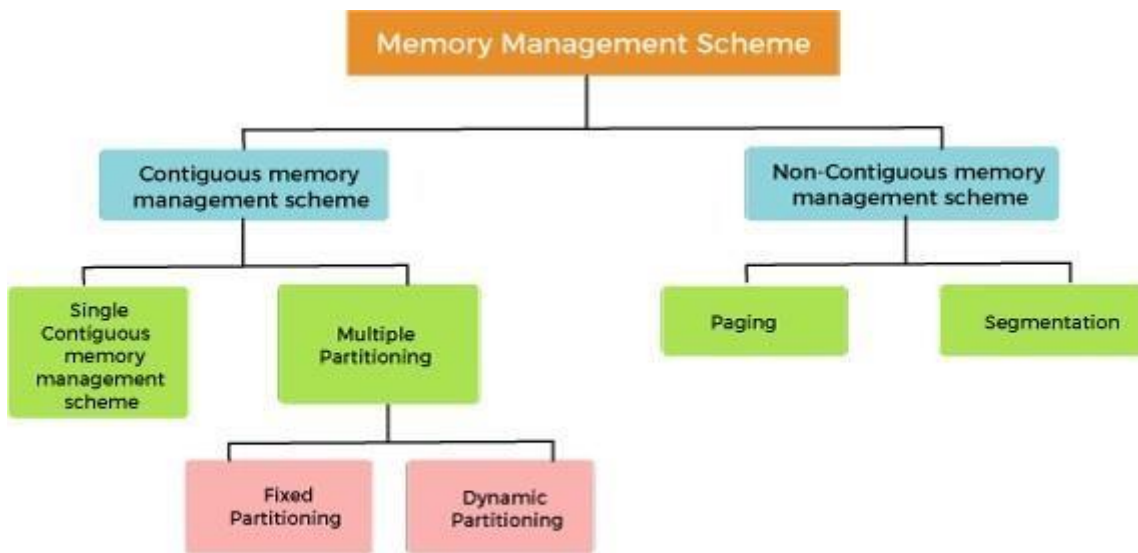
Following are the important roles in a computer system:

- Memory manager is used to keep track of the status of memory locations, whether it is free or allocated. It addresses primary memory by providing abstractions so that software perceives a large memory is allocated to it.
- Memory manager permits computers with a small amount of main memory to execute programs larger than the size or amount of available memory. It does this by moving information back and forth between primary memory and secondary memory by using the concept of swapping.
- The memory manager is responsible for protecting the memory allocated to each process from being corrupted by another process. If this is not ensured, then the system may exhibit unpredictable behavior.
- Memory managers should enable sharing of memory space between processes. Thus, two programs can reside at the same memory location although at different times.

Memory management Techniques:

The Memory management Techniques can be classified into following main categories:

- Contiguous memory management schemes
- Non-Contiguous memory management schemes



Classification of memory management schemes

Contiguous memory management schemes:

In a Contiguous memory management scheme, each program occupies a single contiguous block of storage locations, i.e., a set of memory locations with consecutive addresses.

Single contiguous memory management schemes:

The Single contiguous memory management scheme is the simplest memory management scheme used in the earliest generation of computer systems. In this scheme, the main memory is divided into two contiguous areas or partitions. The operating systems reside permanently in one partition, generally at the lower memory, and the user process is loaded into the other partition.

Advantages of Single contiguous memory management schemes:

- Simple to implement.
- Easy to manage and design.
- In a Single contiguous memory management scheme, once a process is loaded, it is given full processor's time, and no other processor will interrupt it.

Disadvantages of Single contiguous memory management schemes:

- Wastage of memory space due to unused memory as the process is unlikely to use all the available memory space.
- The CPU remains idle, waiting for the disk to load the binary image into the main memory.
- It can not be executed if the program is too large to fit the entire available main memory space.
- It does not support multiprogramming, i.e., it cannot handle multiple programs simultaneously.

Multiple Partitioning:

The single Contiguous memory management scheme is inefficient as it limits computers to execute only one program at a time resulting in wastage in memory space and CPU time. The problem of inefficient CPU use can be overcome using multiprogramming that allows more than one program to run concurrently. To switch between two processes, the operating systems need to load both processes into the main memory. The

operating system needs to divide the available main memory into multiple parts to load multiple processes into the main memory. Thus multiple processes can reside in the main memory simultaneously.

- Fixed Partitioning
- Dynamic Partitioning

Fixed Partitioning

The main memory is divided into several fixed-sized partitions in a fixed partition memory management scheme or static partitioning. These partitions can be of the same size or different sizes. Each partition can hold a single process. The number of partitions determines the degree of multiprogramming, i.e., the maximum number of processes in memory. These partitions are made at the time of system generation and remain fixed after that.

Advantages of Fixed Partitioning memory management schemes:

- Simple to implement.
- Easy to manage and design.

Disadvantages of Fixed Partitioning memory management schemes:

- This scheme suffers from internal fragmentation.
- The number of partitions is specified at the time of system

generation. Dynamic Partitioning

The dynamic partitioning was designed to overcome the problems of a fixed partitioning scheme. In a dynamic partitioning scheme, each process occupies only as much memory as they require when loaded for processing. Requested processes are allocated memory until the entire physical memory is exhausted or the remaining space is insufficient to hold the requesting process. In this scheme the partitions used are of variable size, and the number of partitions is not defined at the system generation time.

Advantages of Dynamic Partitioning memory management schemes:

- Simple to implement.
- Easy to manage and design.

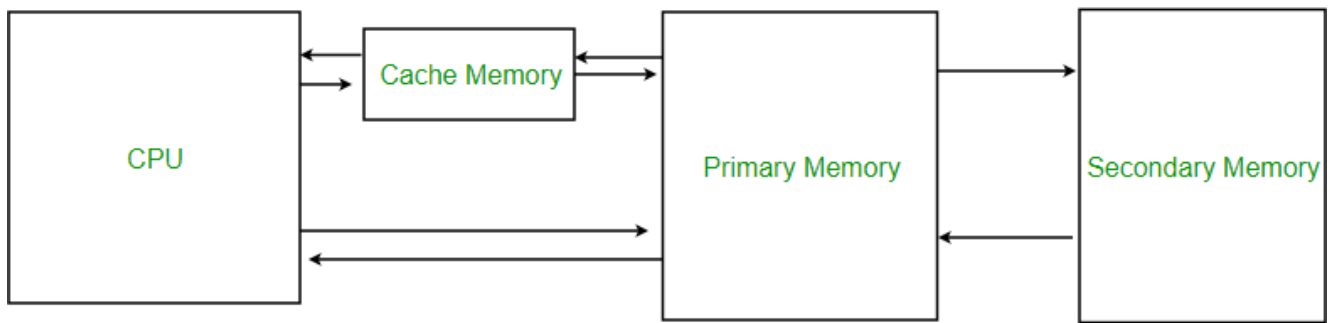
Disadvantages of Dynamic Partitioning memory management schemes:

- This scheme also suffers from internal fragmentation.
- The number of partitions is specified at the time of system segmentation.

Non-Contiguous memory management schemes:

In a Non-Contiguous memory management scheme, the program is divided into different blocks and loaded at different portions of the memory that need not necessarily be adjacent to one another. This scheme can be classified depending upon the size of blocks and whether the blocks reside in the main memory or not.

Cache memory



Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.

Levels of memory:

- **Level 1 or Register –**

It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.

- **Level 2 or Cache memory –**

It is the fastest memory which has faster access time where data is temporarily stored for faster access.

- **Level 3 or Main Memory –**

It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.

- **Level 4 or Secondary Memory –**

It is external memory which is not as fast as main memory but data stays permanently in this memory.

Cache Performance:

When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from cache
- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio**.

Hit ratio = hit / (hit + miss) = no. of hits/total accesses

We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty, and reduce the time to hit in the cache.

Cache Measures

- **Cache:** Cache is small, fast storage used to improve average access time to slow memory. It applied whenever buffering is employed to reuse commonly occurring items, i.e. file caches, name caches, and so on.
- **Cache Hit:** CPU finds a requested data item in the cache.
- **Cache Miss:** The item is not in the cache at access.
- Block is a fixed size collection of data, retrieved from memory and placed into the cache.
- **Advantage of Temporal Locality:** If access data from slower memory, move it to faster memory. If data in faster memory is unused recently, move it to slower memory.
- **Advantage of Spatial Locality:** If need to move a word from slower to faster memory, move adjacent words at same time.
- **Hit Rate (Hit Ratio):** Fraction of accesses that are hits at a given level of the hierarchy.
- **Hit Time:** Time required accessing a level of the hierarchy, including time to determine whether access is a hit or miss.
- **Miss Rate (Miss Ratio):** Fraction of accesses that are misses at a given level.
- **Miss Penalty:** Extra time required to fetch a block into some level from the next level down.
- The address space is usually broken into fixed size blocks, called pages. At each time, each page resides either in main memory or on disk.
- Average memory access time is a useful measure to evaluate the performance of a memory-hierarchy configuration.

Average Memory Access Time = Memory Hit Time + Memory Miss Rate x Miss Penalty

Cache Mapping

3. *Discuss in detail about various cache mapping techniques.*

- When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.
 - ✓ If the processor finds that the memory location is in the cache, a cache hit has occurred and data is read from cache
 - ✓ If the processor does not find the memory location in the cache, a cache miss has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, and then the request is fulfilled from the contents of the cache.
- The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio**.

Hit Ratio = Hit / (Hit + Miss) = No. of Hits / Total Accesses

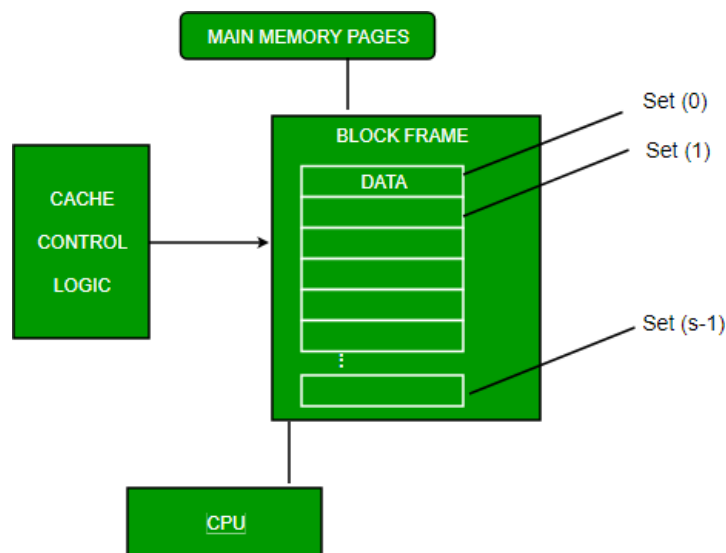
- We can improve Cache performance using higher cache block size, higher associativity, reduce miss rate, reduce miss penalty and reduce the time to hit in the cache.

Cache Mapping

- Cache memory mapping is the way in which we map or organize data in cache memory, this is done for efficiently storing the data which then helps in easy retrieval of the same.
- The three different types of mapping used for the purpose of cache memory are as follow,
 - ✓ Direct Mapping
 - ✓ Associative Mapping
 - ✓ Set-Associative Mapping

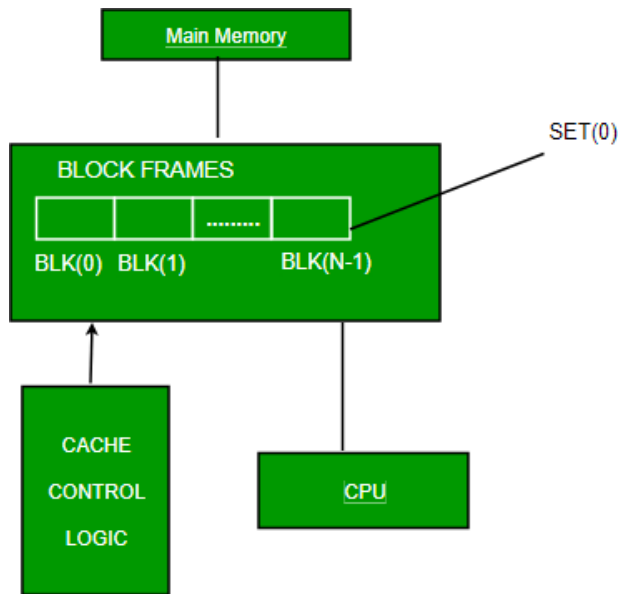
Direct Mapping:

- In direct mapping, assigned each memory block to a specific line in the cache.
- If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed.
- An address space is split into two parts index field and tag field.
- The cache is used to store the tag field whereas the rest is stored in the main memory.
- Direct mapping's performance is directly proportional to the Hit ratio.



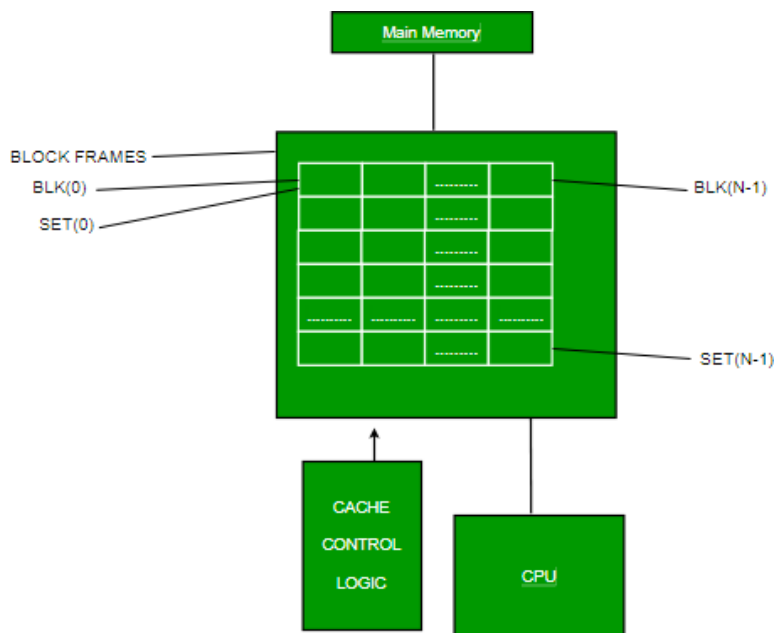
Associative Mapping:

- In this type of mapping, the associative memory is used to store content and addresses both of the memory word. Any block can go into any line of the cache.
- This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits.
- This enables the placement of the any word at any place in the cache memory.
- It is considered to be the fastest and the most flexible mapping form.



Set-Associative Mapping:

- This form of mapping is an enhanced form of the direct mapping where the drawbacks of direct mapping are removed.
- Set associative addresses the problem of possible thrashing in the direct mapping method.
- It does this by saying that instead of having exactly one line that a block can map to in the cache; we will group a few lines together creating a *set*.
- Then a block in memory can map to any one of the lines of a specific set.
- Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address.
- Set associative cache mapping combines the best of direct and associative cache mapping techniques.



Uses of Cache

- Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.

- The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

Types of Cache

- **Primary Cache** – A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.
- **Secondary Cache** – secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.

Locality of Reference

- Since size of cache memory is less as compared to main memory.
- So to check which part of main memory should be given priority and loaded in cache is decided based on locality of reference.

Types of Locality of Reference

- **Spatial Locality of reference** – this says that there is chance that element will be present in the close proximity to the reference point and next time if again searched then more close proximity to the point of reference.
- **Temporal Locality of reference** – In this Least recently used algorithm will be used. Whenever there is page fault occurs within word will not only load word in main memory but complete page fault will be loaded because spatial locality of reference rule says that if you are referring any word next word will be referred in its register that's why we load complete page table so complete block will be loaded.

Cache replacement Techniques:

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when a new page comes in.

Page Fault: A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of a page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

Page Replacement Algorithms:

1. First In First Out (FIFO): This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example 1: Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames. Find the number of page faults.

**Page
reference**

1, 3, 0, 3, 5, 6, 3

1	3	0	3	5	6	3
		0	0	0	0	3
	3	3	3	3	6	6
1	1	1	1	5	5	5
Miss	Miss	Miss	Hit	Miss	Miss	Miss

Total Page Fault = 6

Initially, all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots → **3 Page Faults**. when 3 comes, it is already in memory so → **0 Page Faults**. Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. → **1 Page Fault**. 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 → **1 Page Fault**. Finally, when 3 come it is not available so it replaces 0 **1 page fault**.

Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference strings 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4, and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10-page faults.

2. Optimal Page replacement: In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example-2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frame. Find number of page fault.

Page reference

7,0,1,2,0,3,0,4,2,3,0,3,2,3

No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit

Total Page Fault = 6

Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots → 4 Page faults
 0 is already there so → 0 Page fault. when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future. → 1 Page fault. 0 is already there so → 0 Page fault. 4 will take place of 1 → 1 Page Fault.

Now for the further page reference string → 0 Page fault because they are already available in the memory.

Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

3. Least Recently Used: In this algorithm, page will be replaced which is least recently used.

Example-3: Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames. Find number of page faults.

Page reference

7,0,1,2,0,3,0,4,2,3,0,3,2,3

No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3
Miss	Miss	Miss	Miss	Hit	Miss	Hit	Miss	Hit	Hit	Hit	Hit	Hit	Hit

Total Page Fault = 6

Here LRU has same number of page fault as optimal but it may differ according to question.

Initially, all slots are empty, so when 7 0 1 2 are allocated to the empty slots → 4 Page faults
 0 is already there so → 0 Page fault. when 3 came it will take the place of 7 because it is least recently

used → **1 Page fault** 0 is already in memory so → **0 Page fault**. 4 will take place of 1 → **1 Page Fault** Now for the further page reference string → **0 Page fault** because they are already available in the memory.

4. Most Recently Used (MRU): In this algorithm, page will be replaced which has been used recently. Belady's anomaly can occur in this algorithm.

4. Explain in detail about virtual memory with an example.(Nov/Dec 2019) (Nov/Dec 2021) or Discuss the concept of virtual memory and explain how a virtual memory system is implemented, pointing out the hardware and software support. (Nov/Dec 2017)Nov/Dec 2020.

VIRTUAL MEMORY

- Virtual memory divides physical memory into blocks (called page or segment) and allocates them to different processes.
- With virtual memory, the CPU produces virtual addresses that are translated by a combination of HW and SW to physical addresses, which accesses main memory.
- The process is called memory mapping or address translation.
- Today, the two memory-hierarchy levels controlled by virtual memory are DRAMs and magnetic disks.
- Virtual Memory manages the two levels of the memory hierarchy represented by main memory and secondary storage.
- Figure below shows the mapping of virtual memory to physical memory for a program with four pages.

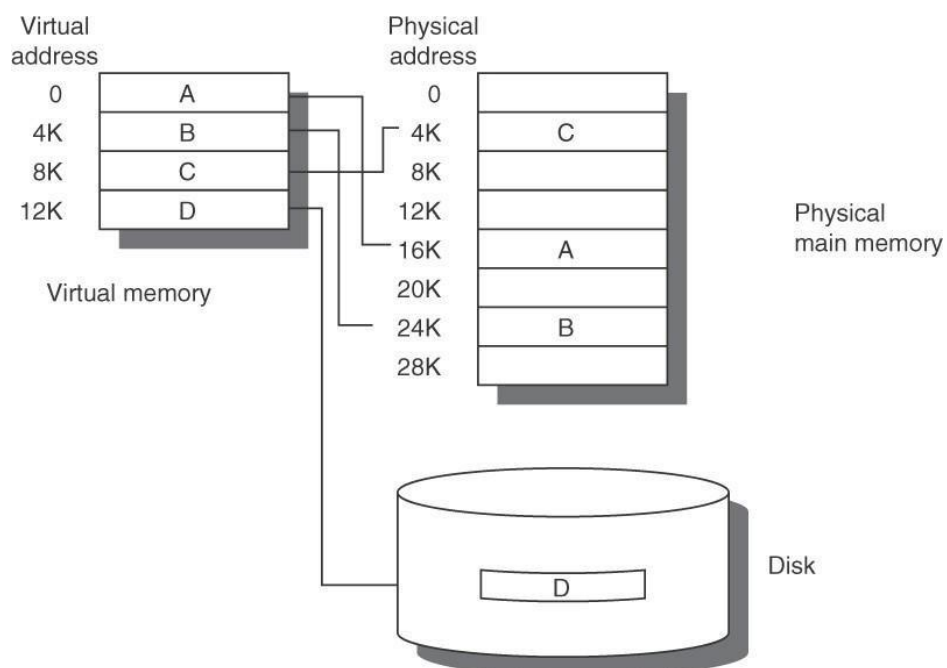
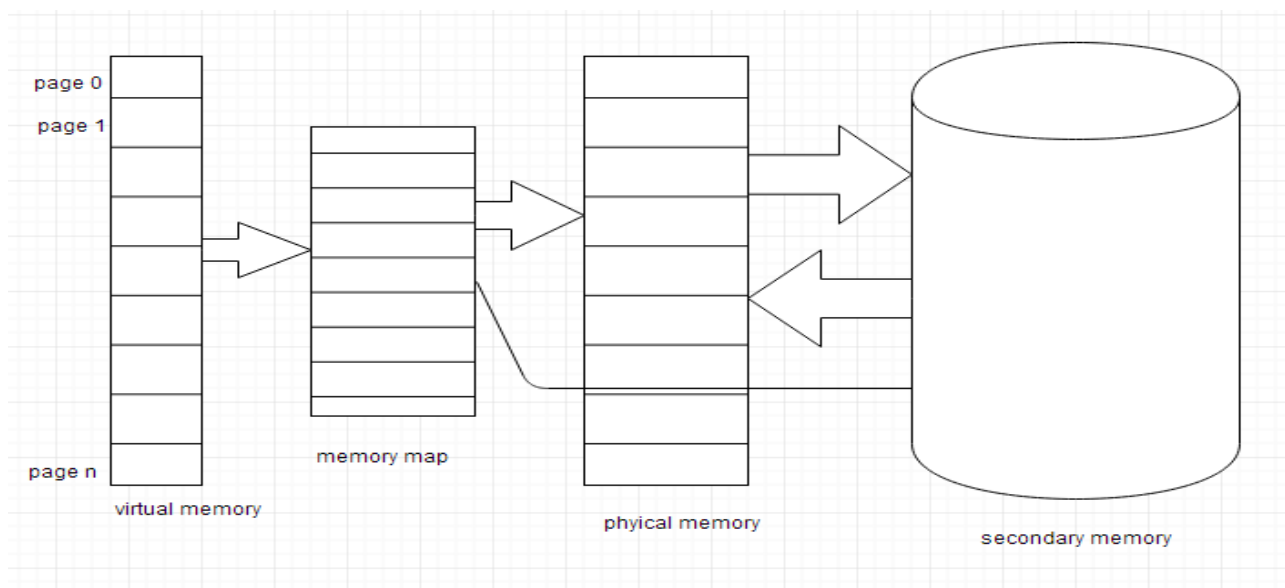


Figure: Virtual Memory Space

- Virtual memory is the separation of logical memory from physical memory.

- This separation provides large virtual memory for programmers when only small physical memory is available.
- Virtual memory is a memory management capability of an OS that uses hardware and software to allow a computer to compensate for physical memory shortages by temporarily transferring data from random access memory (RAM) to disk storage.
- Virtual address space is increased using active memory in RAM and inactive memory in hard disk drives (HDDs) to form contiguous addresses that hold both the application and its data.
- Computers have a finite amount of RAM so memory can run out, especially when multiple programs run at the same time.
- A system using virtual memory can load larger programs or multiple programs running at the same time, allowing each one to operate as if it has infinite memory and without having to purchase more RAM.
- As part of the process of copying virtual memory into physical memory, the OS divides memory into page files or swap files that contain a fixed number of addresses.
- Each page is stored on a disk and when the page is needed, the OS copies it from the disk to main memory and translates the virtual addresses into real addresses.



Pros and Cons of using Virtual Memory

- Among the primary benefits of virtual memory is its ability to handle twice as many addresses as main memory.
- It uses software to consume more memory by using the HDD as temporary storage while memory management units translate virtual memory addresses to physical addresses via the central processing unit.
- Programs use virtual addresses to store instructions and data; when a program is executed, the virtual addresses are converted into actual memory addresses.

5. Discuss the concept of Programmed I/O. Discuss about Programmed I/Os associated with computers. (Apr/May 2018)

Programmed I/O

- If I/O operations are completely controlled by the CPU, the computer is said to be using **programmed I/O**. In this case, the CPU executes programs that initiate, direct and terminate the I/O operations.
- If a part of the main memory address space is assigned to I/O ports, then such systems are called as **Memory-Mapped I/O systems**.
- In **I/O-mapped I/O** systems, the memory and I/O address space are separate. Similarly the control lines used for activating memory and I/O devices are also different. Two sets of control lines are available. READ M and WRITE M are related with memory and READ I/O and WRITE I/O are related with I/O devices.

S.No.	Parameter	Memory-mapped I/O	I/O-mapped I/O
1.	Address space	Memory and I/O devices share the entire address space	Memory and I/O devices have separate address space
2.	Hardware	No additional hardware required	Additional hardware required
3.	Implementation	Easy to implement	Difficult to implement
4.	Address	Same address cannot be used to refer both memory and I/O device.	Same address can be used to refer both memory and I/O device.
5.	Control lines	Memory control lines are used to control I/O devices.	Different set of control lines are used to control memory and I/O.
6.	Control lines used	The control lines are: READ, WRITE	The control lines are: READ M, WRITE M, READ I/O, WRITE I/O

I/O instructions

Two I/O instructions are used to implement programmed I/O.

- **IN:** The instruction IN X causes a word to be transferred from I/O port X to the accumulator register A.
- **OUT:** The instruction OUT X transfer a word from the accumulator register A to the I/O port X.

Limitations of programmed I/O

The programmed I/O method has two limitations:

- The speed of the CPU is reduced due to low speed I/O devices.
- Most of the CPU time is wasted

6. Describe the DMA controller in a computer system with a neat block diagram. Explain mechanism Direct Memory Access. (Nov/Dec2012, 2013, 2015, 2016) (Apr / May 2016, 2017, Nov /Dec 2011) (Nov/Dec 2018).With a neat sketch explain the working principle of DMA. (Apr/May 2019)

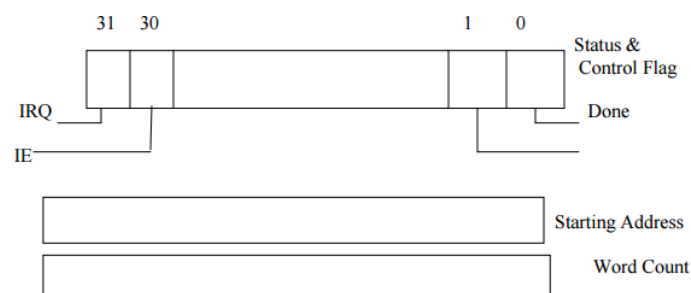
DIRECT MEMORY ACCESS

- A special control unit may be provided to allow the transfer of large block of data at high speed directly between the external device and main memory, without continuous intervention by the processor. This approach is called DMA.
- DMA transfers are performed by a control circuit called the **DMA Controller**.

To initiate the transfer of a block of words, the processor sends,

- i) Starting address
- ii) Number of words in the block
- iii) Direction of transfer.

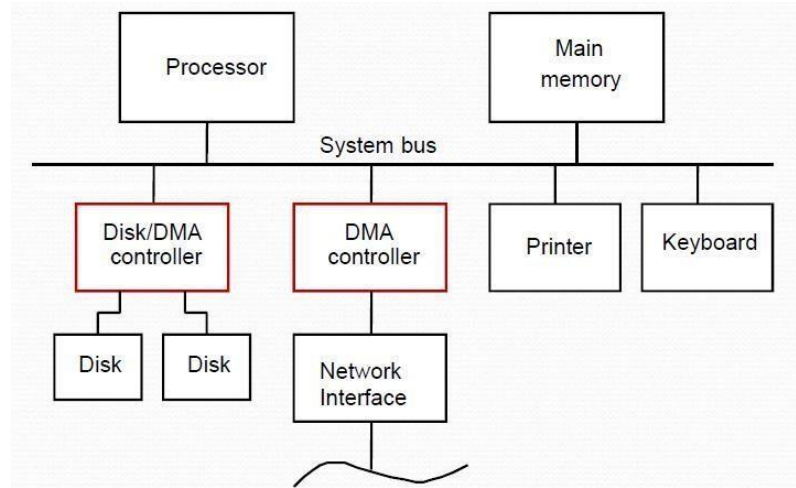
- When a block of data is transferred, the DMA controller increment the memory address for successive words and keep track of number of words and it also informs the processor by raising an interrupt signal.



- While DMA control is taking place, the program requested the transfer cannot continue and the processor can be used to execute another program.
- After DMA transfer is completed, the processor returns to the program that requested the transfer.

R/W->Determines the direction of transfer

- When **R/W =1**, DMA controller read data from memory to I/O device.
- **R/W =0**, DMA controller perform write operation.
- **Done Flag=1**, the controller has completed transferring a block of data and is ready to receive another command.
- **IE=1**, it causes the controller to raise an interrupt (interrupt Enabled) after it has completed transferring the block of data.
- **IRQ=1**, it indicates that the controller has requested an interrupt.



- A DMA controller connects a high speed network to the computer bus, and the disk controller for two disks also has DMA capability and it provides two DMA channels.
- To start a DMA transfer of a block of data from main memory to one of the disks, the program write's the address and the word count information into the registers of the corresponding channel of the disk controller.
- When DMA transfer is completed, it will be recorded in status and control registers of the DMA channel (ie) Done bit=IRQ=IE=1.

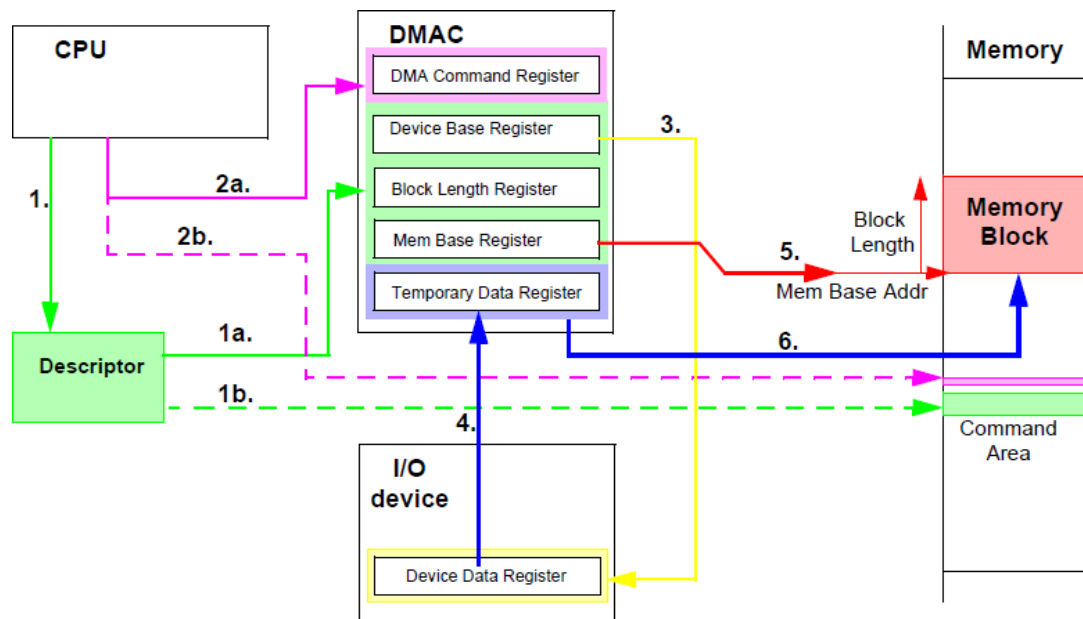
DMA Operations: May 2009

A lot of different operating modes exist for DMACs. The simplest one is the single block transfer copying a block of data from a device to memory. For the more complex operations please refer to the literature.

Here, only a short list of operating modes is given:

- Single block transfer
- Chained block transfers
- Linked block transfers
- Fly-by transfers

All these operations normally access the block of data in a linear sequence. Nevertheless, there are more usefull access functions possible, as there are: constant stride, constant stride with offset, incremental stride.



Execution of a DMA-operation (single block transfer)

- The CPU prepares the DMA-operation by the construction of a descriptor (1), containing all necessary information for the DMAC to independently perform the DMA-operation (offload engine for data transfer).
- It initializes the operation by writing a command to a register in the DMAC (2a) or to a special assigned memory area (command area), where the DMAC can poll for the command and/or the descriptor (2b). Then the DMAC addresses the device data register (3) and read the data into a temporary data register (4).
- In another bus transfer cycle, it addresses the memory block (5) and writes the data from the temporary data register to the memory block (6).

Cycle Stealing:

- Requests by DMA devices for using the bus are having higher priority than processor requests.
- Top priority is given to high speed peripherals such as, Disk High speed Network Interface and Graphics display device.
- Since the processor originates most memory access cycles, the DMA controller can be said to steal the memory cycles from the processor. This interviewing technique is called **Cycle stealing**.
- **Burst Mode:** The DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as **Burst/Block Mode**.
- **Bus Master:** The device that is allowed to initiate data transfers on the bus at any given time is called the **bus master**.

BUS ARBITRATION:

7. Explain in detail about the Bus Arbitration techniques in DMA. (Nov 2011, 2012, 2014) Apr / May 2011, 2017, May 2013

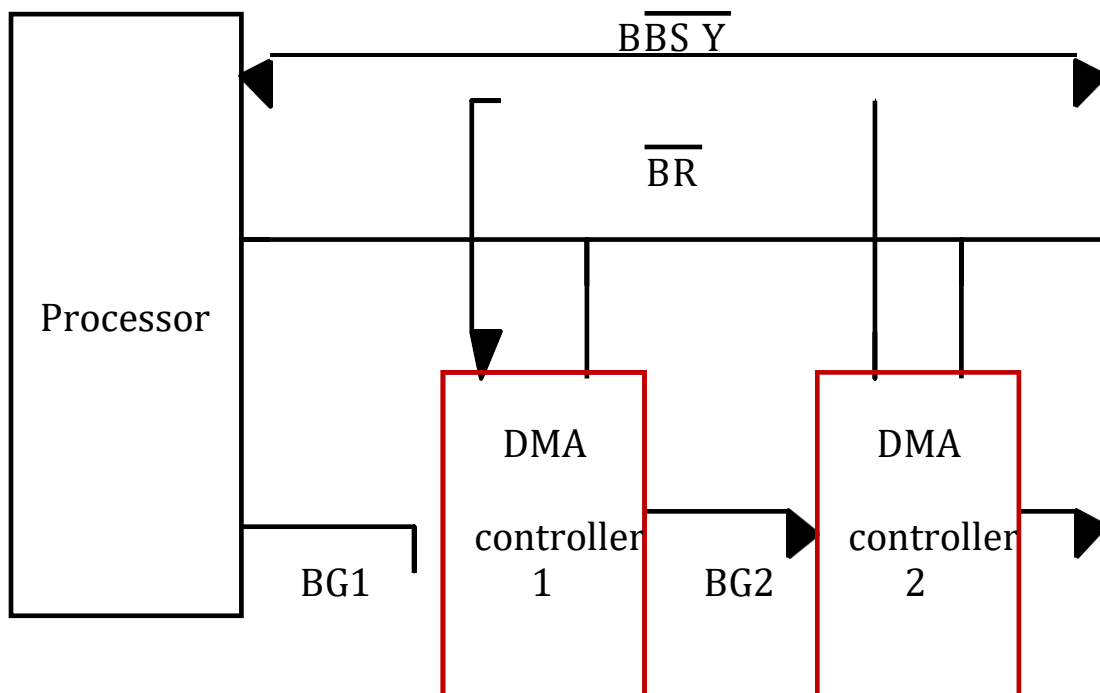
- **Bus Arbitration:** It is the process by which the next device to become the bus master is selected and the bus mastership is transferred to it.

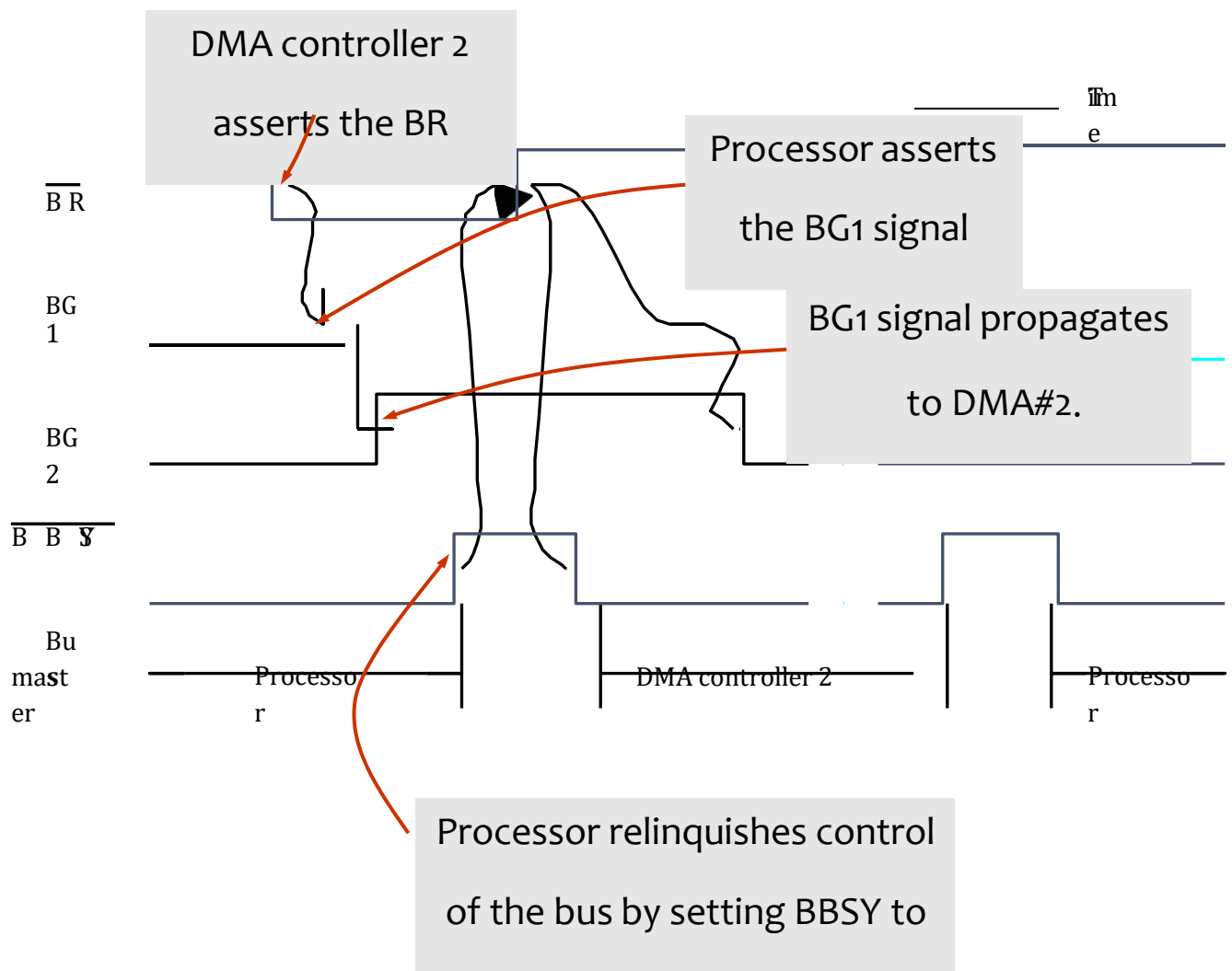
Types: There are 2 approaches to bus arbitration. They are,

- Centralized arbitration (A single bus arbiter performs arbitration)
- Distributed arbitration (all devices participate in the selection of next bus master).

Centralized Arbitration:

- Here the processor is the bus master and it may grants bus mastership to one of its DMA controller.
- A DMA controller indicates that it needs to become the bus master by activating the Bus Request line (BR) which is an open drain line.
- The signal on BR is the logical OR of the bus request from all devices connected to it. When BR is activated the processor activates the Bus Grant Signal (BGI) and indicated the DMA controller that they may use the bus when it becomes free.
- This signal is connected to all devices using a daisy chain arrangement.
- If DMA requests the bus, it blocks the propagation of Grant Signal to other devices and it indicates to all devices that it is using the bus by activating open collector line, Bus Busy (BBSY).



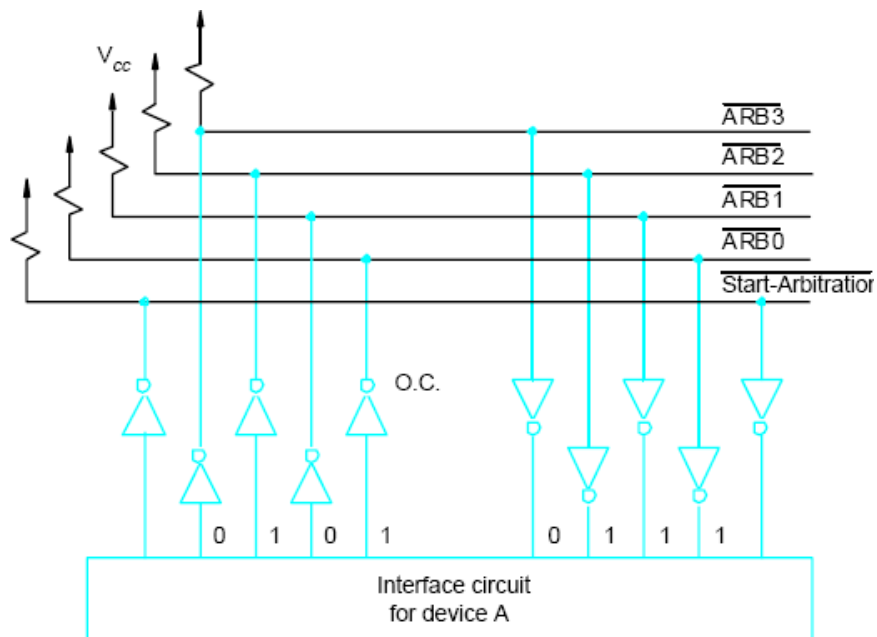


Sequence of signals during transfer of bus mastership for the devices

- The timing diagram shows the sequence of events for the devices connected to the processor is shown.
- DMA controller 2 requests and acquires bus mastership and later releases the bus.
- During its tenure as bus master, it may perform one or more data transfer.
- After it releases the bus, the processor resumes bus mastership.

Distributed Arbitration

- It means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process.



- Each device on the bus is assigned a 4 bit id. When one or more devices request the bus, they assert the Start-Arbitration signal & place their 4 bit ID number on four open collector lines, ARB0 to ARB3.
- A winner is selected as a result of the interaction among the signals transmitted over these lines.
- The net outcome is that the code on the four lines represents the request that has the highest ID number.
- The drivers are of open collector type. Hence, if the i/p to one driver is equal to 1, the i/p to another driver connected to the same bus line is equal to 0' (ie. bus is in low-voltage state).
- **Eg:** Assume two devices A & B have their ID 5 (0101), 6(0110) and their code is 0111.
- Each device compares the pattern on the arbitration line to its own ID starting from MSB.
- If it detects a difference at any bit position, it disables the drivers at that bit position. It does this by placing 0' at the i/p of these drivers.
- A' detects a difference in line ARB1; hence it disables the drivers on lines ARB1 & ARB0. This causes the pattern on the arbitration line to change to 0110 which means that B' has won the contention.

INPUT DEVICES:

8. Explain in detail about input devices with an example.

Input Devices

The Input Devices are the hardware that is used to transfer transfers input to the computer. The data can be in the form of text, graphics, sound, and text. Output device display data from the memory of the computer.

Output can be text, numeric data, line, polygon, and other objects.

These Devices include:

1. Keyboard

2. Mouse
 3. Trackball
 4. Spaceball
 5. Joystick
 6. Light Pen
 7. Digitizer
 8. Touch Panels
 9. Voice Recognition
 10. Image Scanner
-

Keyboard:

The most commonly used input device is a keyboard. The data is entered by pressing the set of keys. All keys are labeled. A keyboard with 101 keys is called a QWERTY keyboard.

The keyboard has alphabetic as well as numeric keys. Some special keys are also available.

1. **Numeric Keys:** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
2. **Alphabetic keys:** a to z (lower case), A to Z (upper case)
3. **Special Control keys:** Ctrl, Shift, Alt
4. **Special Symbol Keys:** ; , " ? @ ~ ? :
5. **Cursor Control Keys:** ↑ → ← ↓
6. **Function Keys:** F1 F2 F3... F9.
7. **Numeric Keyboard:** It is on the right-hand side of the keyboard and used for fast entry of numeric data.

Function of Keyboard:

1. Alphanumeric Keyboards are used in CAD. (Computer Aided Drafting)
2. Keyboards are available with special features line screen co-ordinates entry, Menu selection or graphics functions, etc.
3. Special purpose keyboards are available having buttons, dials, and switches. Dials are used to enter scalar values. Dials also enter real numbers. Buttons and switches are used to enter predefined function values.

Advantage:

1. Suitable for entering numeric data.
2. Function keys are a fast and effective method of using commands, with fewer errors.

Disadvantage:

1. Keyboard is not suitable for graphics input.

Mouse:

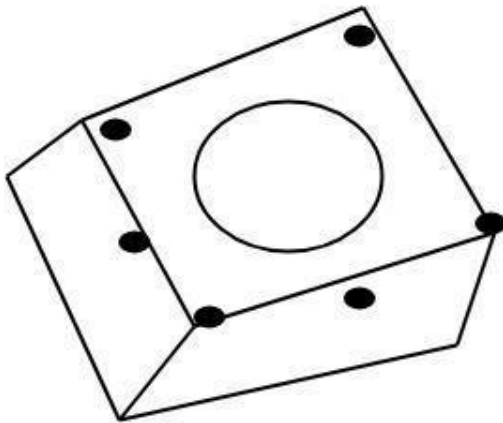
A Mouse is a pointing device and used to position the pointer on the screen. It is a small palm size box. There are two or three depression switches on the top. The movement of the mouse along the x-axis helps in the horizontal movement of the cursor and the movement along the y-axis helps in the vertical movement of the cursor on the screen. The mouse cannot be used to enter text. Therefore, they are used in conjunction with a keyboard.

Advantage:

1. Easy to use
2. Not very expensive

Trackball

It is a pointing device. It is similar to a mouse. This is mainly used in notebook or laptop computer, instead of a mouse. This is a ball which is half inserted, and by changing fingers on the ball, the pointer can be



TrackBall

moved.

Advantage:

1. Trackball is stationary, so it does not require much space to use it.
2. Compact Size

Spaceball:

It is similar to trackball, but it can move in six directions where trackball can move in two directions only. The movement is recorded by the strain gauge. Strain gauge is applied with pressure. It can be pushed and pulled in various directions. The ball has a diameter around 7.5 cm. The ball is mounted in the base using rollers. One-third of the ball is an inside box, the rest is outside.

Applications:

1. It is used for three-dimensional positioning of the object.
2. It is used to select various functions in the field of virtual reality.
3. It is applicable in CAD applications.
4. Animation is also done using spaceball.

5. It is used in the area of simulation and modeling.

Joystick:

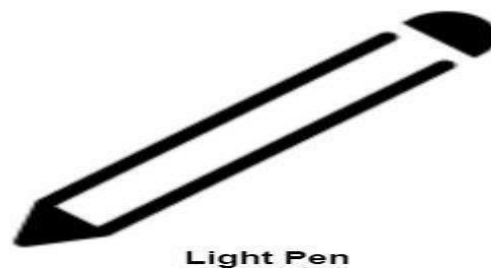
A Joystick is also a pointing device which is used to change cursor position on a monitor screen. Joystick is a stick having a spherical ball as its both lower and upper ends as shown in fig. The lower spherical ball moves in a socket. The joystick can be changed in all four directions. The function of a joystick is similar to that of the mouse. It is mainly used in Computer Aided Designing (CAD) and playing computer games.



When the wave signals are interrupted by some contact with the screen, that located is recorded. Touch screens have long been used in military applications.

Light Pen

Light Pen (similar to the pen) is a pointing device which is used to select a displayed menu item or draw pictures on the monitor screen. It consists of a photocell and an optical system placed in a small tube. When its tip is moved over the monitor screen, and pen button is pressed, its photocell sensing element detects the screen location and sends the corresponding signals to the CPU.



Uses:

1. Light Pens can be used as input coordinate positions by providing necessary arrangements.
2. If background color or intensity, a light pen can be used as a locator.
3. It is used as a standard pick device with many graphics system.
4. It can be used as stroke input devices.
5. It can be used as valuator

Digitizers:

The digitizer is an operator input device, which contains a large, smooth board (the appearance is similar to the mechanical drawing board) & an electronic tracking device, which can be changed over the surface to follow existing lines. The electronic tracking device contains a switch for the user to record the desire x & y

coordinate positions. The coordinates can be entered into the computer memory or stored on an off-line storage medium such as magnetic tape.



Digitizer

Advantages:

1. Drawing can easily be changed.
2. It provides the capability of interactive graphics.

Disadvantages:

1. Costly
2. Suitable only for applications which required high-resolution graphics.

Touch Panels:

Touch Panels is a type of display screen that has a touch-sensitive transparent panel covering the screen. A touch screen registers input when a finger or other object comes in contact with the screen.

When the wave signals are interrupted by some contact with the screen, that located is recorded. Touch screens have long been used in military applications.

OUTPUT DEVICES:

9. Explain in detail about Output devices with an example.

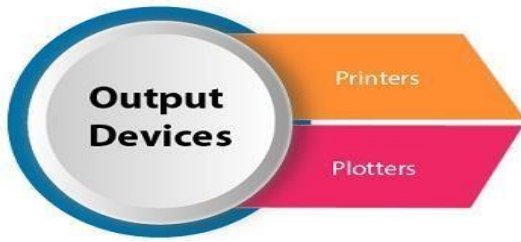
It is an electromechanical device, which accepts data from a computer and translates them into form understand by users.

Following are Output Devices:

1. [Printers](#)
2. [Plotters](#)

Printers:

Printer is the most important output device, which is used to print data on paper.



Types of Printers:

1. Impact Printers: The printers that print the characters by striking against the ribbon and onto the papers are known as Impact Printers.

These Printers are of two types:

1. Character Printers
2. Line Printers

2. Non-Impact Printers: The printers that print the characters without striking against the ribbon and onto the papers are called Non-Impact Printers. These printers print a complete page at a time, therefore, also known as Page Printers.

Page Printers are of two types:

1. Laser Printers
2. Inkjet Printers

Laser Printers:

These are non-impact page printers. They use laser lights to produce the dots needed to form the characters to be printed on a page & hence the name laser printers.

The output is generated in the following steps:

Step1: The bits of data sent by processing unit act as triggers to turn the laser beam on & off.

Step2: The output device has a drum which is cleared & is given a positive electric charge. To print a page the modulated laser beam passing from the laser scans back & forth the surface of the drum. The positive electric charge on the drum is stored on just those parts of the drum surface which are exposed to the laser beam create the difference in electric which charges on the exposed drum surface.

Step3: The laser exposed parts of the drum attract an ink powder known as toner.

Step4: The attracted ink powder is transferred to paper.

Step5: The ink particles are permanently fixed to the paper by using either heat or pressure technique.

Step6: The drum rotates back to the cleaner where a rubber blade cleans off the excess ink & prepares the drum to print the next page.

Liquid Crystal Displays (LCDs)

LCD Display Monitor

The flat-panel display refers to a class of video devices that have reduced volume, weight and power requirement in comparison to the CRT. You can hang them on walls or wear them on your wrists. Current uses of flat-panel displays include calculators, video games, monitors, laptop computer, and graphics display.



The flat-panel display is divided into two categories –

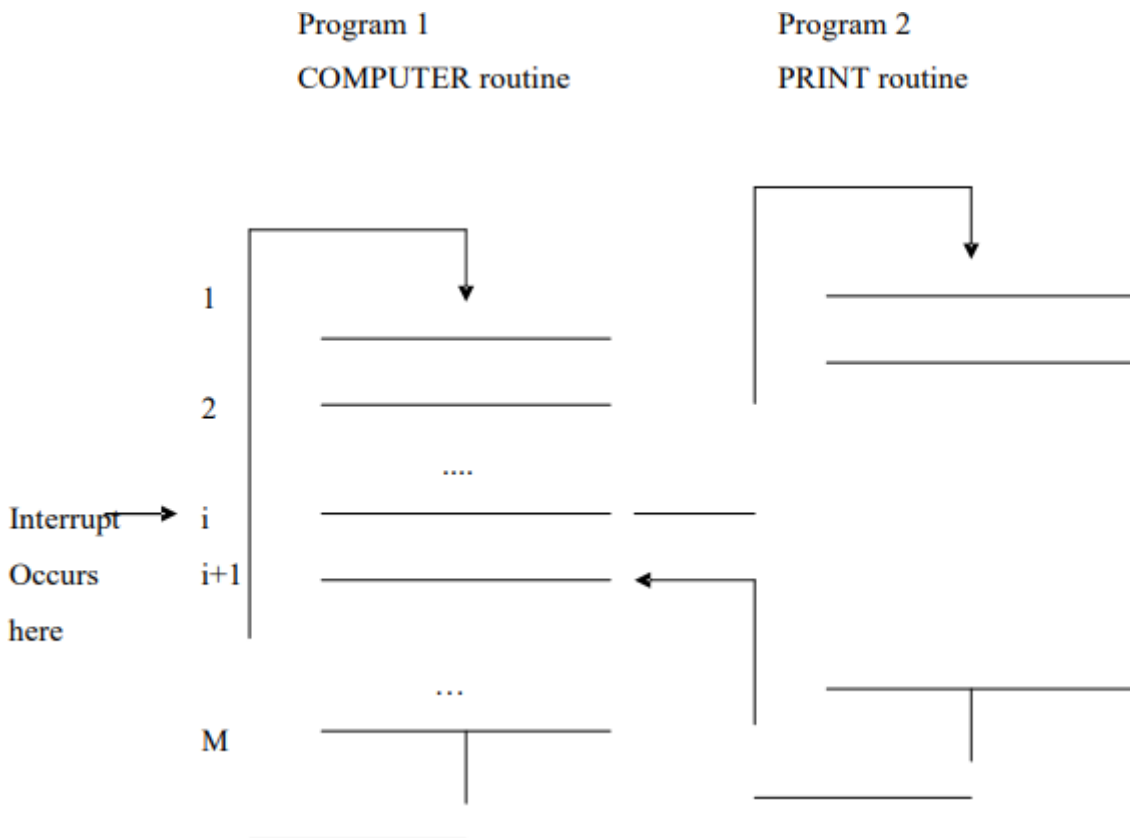
- **Emissive Displays** – Emissive displays are devices that convert electrical energy into light. For example, plasma panel and LED (Light-Emitting Diodes).
- **Non-Emissive Displays** – Non-emissive displays use optical effects to convert sunlight or light from some other source into graphics patterns. For example, LCD (Liquid-Crystal Device).

ACCESSING I/O

10. How to accessing I/O devices to a computer?

- A simple arrangement to connect I/O devices to a computer is to use a single bus arrangement. The bus enables all the devices connected to it to exchange information.
- Typically, it consists of three sets of lines used to carry address, data, and control signals. Each I/O device is assigned a unique set of addresses.
- When the processor places a particular address on the address line, the device that recognizes this address responds to the commands issued on the control lines.
- The processor requests either a read or a write operation, and the requested data are transferred over the data lines, when I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O.
- With memory-mapped I/O, any machine instruction that can access memory can be used to transfer data to or from an I/O device.
- For example, if DATAIN is the address of the input buffer associated with the keyboard, the instruction Move DATAIN, R0 Reads the data from DATAIN and stores them into processor register R0.

- Similarly, the instruction Move R0, DATAOUT Sends the contents of register R0 to location DATAOUT, which may be the output data buffer of a display unit or a printer.
- Most computer systems use memory-mapped I/O. some processors have special In and Out instructions to perform I/O transfers.
- When building a computer system based on these processors, the designer had the option of connecting I/O devices to use the special I/O address space or simply incorporating them as part of the memory address space.
- The I/O devices examine the low-order bits of the address bus to determine whether they should respond. The hardware required to connect an I/O device to the bus.
- The address decoder enables the device to recognize its address when this address appears on the address lines.
- The data register holds the data being transferred to or from the processor. The status register contains information relevant to the operation of the I/O device.
- Both the data and status registers are connected to the data bus and assigned unique addresses.
- The address decoder, the data and status registers, and the control circuitry required to coordinate I/O transfers constitute the device's interface circuit.
- I/O devices operate at speeds that are vastly different from that of the processor. When a human operator is entering characters at a keyboard, the processor is capable of executing millions of instructions between successive character entries.
- An instruction that reads a character from the keyboard should be executed only when a character is available in the input buffer of the keyboard interface.
- Also, we must make sure that an input character is read only once. This example illustrates program-controlled I/O, in which the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device.
- We say that the processor polls the device. There are two other commonly used mechanisms for implementing I/O operations: interrupts and direct memory access.
- In the case of interrupts, synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation.
- Direct memory access is a technique used for high-speed I/O devices. It involves having the device interface transfer data directly to or from the memory, without continuous involvement by the processor.
- The routine executed in response to an interrupt request is called the interrupt service routine, which is the PRINT routine in our example.
- Interrupts bear considerable resemblance to subroutine calls. Assume that an interrupt request arrives during execution of instruction i in figure 1



- The processor first completes execution of instruction i . Then, it loads the program counter with the address of the first instruction of the interrupt-service routine.
- For the time being, let us assume that this address is hardwired in the processor. After execution of the interrupt-service routine, the processor has to come back to instruction $i + 1$.
- Therefore, when an interrupt occurs, the current contents of the PC, which point to instruction $i + 1$, must be put in temporary storage in a known location.
- A Return-from interrupt instruction at the end of the interrupt-service routine reloads the PC from the temporary storage location, causing execution to resume at instruction $i + 1$.
- In many processors, the return address is saved on the processor stack. We should note that as part of handling interrupts, the processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal.
- This may be accomplished by means of a special control signal on the bus. An interrupt-acknowledge signal.
- The execution of an instruction in the interrupt-service routine that accesses a status or data register in the device interface implicitly informs that device that its interrupt request has been recognized.
- So far, treatment of an interrupt-service routine is very similar to that of a subroutine. An important departure from this similarity should be noted.
- A subroutine performs a function required by the program from which it is called. However, the interrupt-service routine may not have anything in common with the program being executed at the time the interrupt request is received.

- In fact, the two programs often belong to different users. Therefore, before starting execution of the interrupt-service routine, any information that may be altered during the execution of that routine must be saved.
- This information must be restored before execution of the interrupt program is resumed. In this way, the original program can continue execution without being affected in any way by the interruption, except for the time delay.
- The information that needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine.
- The task of saving and restoring information can be done automatically by the processor or by program instructions.
- Most modern processors save only the minimum amount of information needed to maintain the registers involves memory transfers that increase the total execution time, and hence represent execution overhead.
- Saving registers also increase the delay between the time an interrupt request is received and the start of execution of the interrupt-service routine. This delay is called interrupt latency.

SERIAL AND PARALLEL INTERFACE

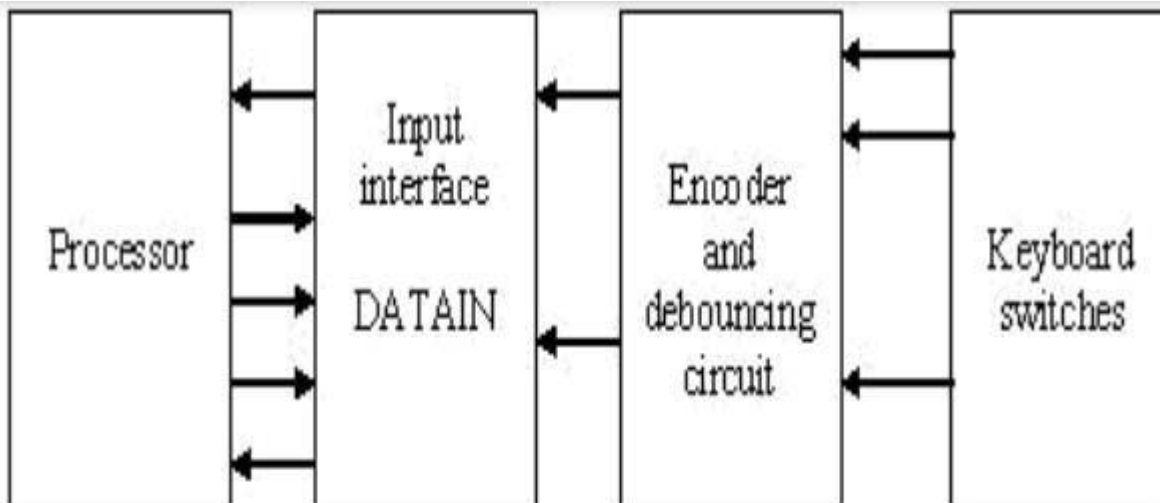
11. Explain about serial and parallel interface.

- An I/O interface consists of the circuitry required to connect an I/O device to a computer bus. On one side of the interface, we have bus signals.
- On the other side, we have a data path with its associated controls to transfer data between the interface and the I/O device – port.
- We have two types: Serial port and Parallel port A parallel port transfers data in the form of a number of bits (8 or 16) simultaneously to or from the device.
- A serial port transmits and receives data one bit at a time. Communication with the bus is the same for both formats.
- The conversion from the parallel to the serial format, and vice versa, takes place inside the interface circuit.
- In parallel port, the connection between the device and the computer uses a multiple-pin connector and a cable with as many wires.
- This arrangement is suitable for devices that are physically close to the computer. In serial port, it is much more convenient and cost-effective where longer cables are needed.
- Typically, the functions of an I/O interface are:
 - ✓ Provides a storage buffer for at least one word of data
 - ✓ Contains status flags that can be accessed by the processor to determine whether the buffer is full or empty
 - ✓ Contains address-decoding circuitry to determine when it is being addressed by the processor
 - ✓ Generates the appropriate timing signals required by the bus control scheme

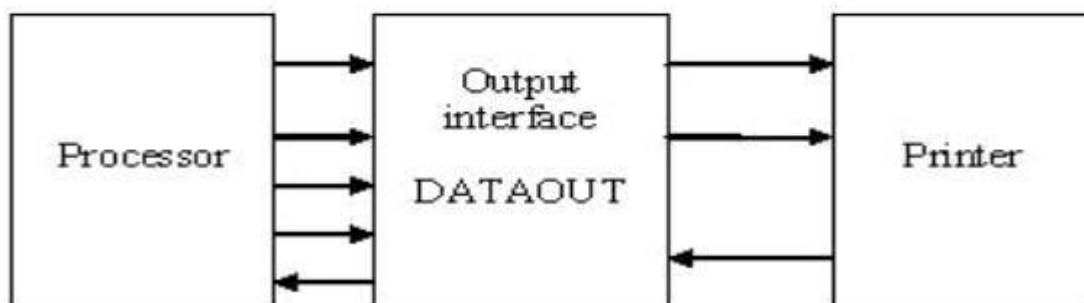
- ✓ Performs any format conversion that may be necessary to transfer data between the bus and the I/O device, such as parallel-serial conversion in the case of a serial port.

Parallel Port

- The hardware components needed for connecting a keyboard to a processor Consider the circuit of input interface which encompasses (as shown in below figure): –Status flag, SIN –R/~W –Master-ready –Address decoder.
- A detailed figure showing the input interface circuit is presented in figure.
- Now, consider the circuit for the status flag (figure 4.30). An edge-triggered D flip-flop is used along with read-data and master-ready signals



Keyboard to processor connection



Printer to processor connection

- The hardware components needed for connecting a printer to a processor are: the circuit of output interface, and –Slave-ready –R/~W –Master-ready –Address decoder –Handshake control.
- The input and output interfaces can be combined into a single interface.
- The general purpose parallel interface circuit that can be configured in a variety of ways.
- For increased flexibility, the circuit makes it possible for some lines to serve as inputs and some lines to serve as outputs, under program control.

Serial Port

- A serial interface circuit involves – Chip and register select, Status and control, Output shift register, DATAOUT, DATAIN, Input shift register and Serial input/output

INTERRUPT

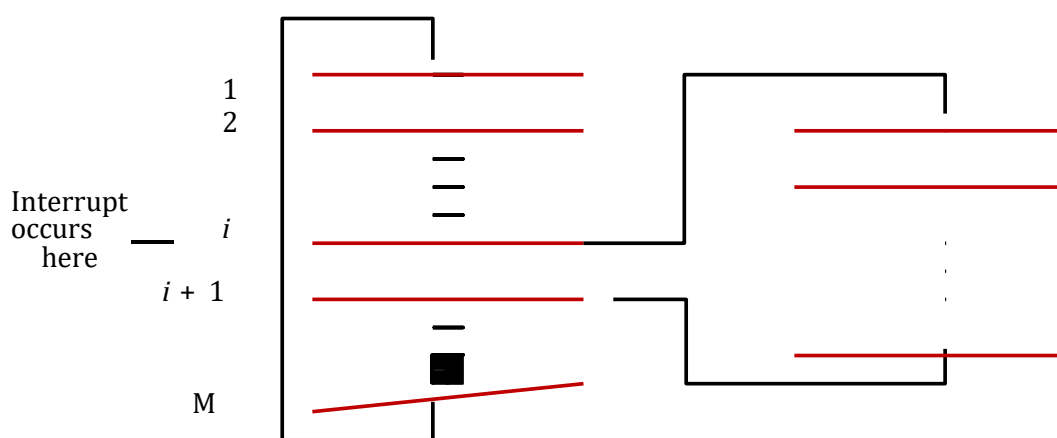
12. What is an interrupt? Explain the different types of interrupts and the different ways of handling the interrupts. Explain Interrupt Handling. (Nov/Dec 2016) (Nov/Dec 2018)Nov/Dec 2020.

INTERRUPTS:

- An interrupt is an event that causes the execution of one program to be suspended and the execution of another program to begin.
- In program-controlled I/O, when the processor continuously monitors the status of the device, the processor will not perform any function.
- An alternate approach would be for the I/O device to alert the processor when it becomes ready. The Interrupt request line will send a hardware signal called the **interrupt signal** to the processor. On receiving this signal, the processor will perform the useful function during the waiting period.
- The routine executed in response to an interrupt request is called **Interrupt Service Routine**. The interrupt resembles the subroutine calls.

Program 1

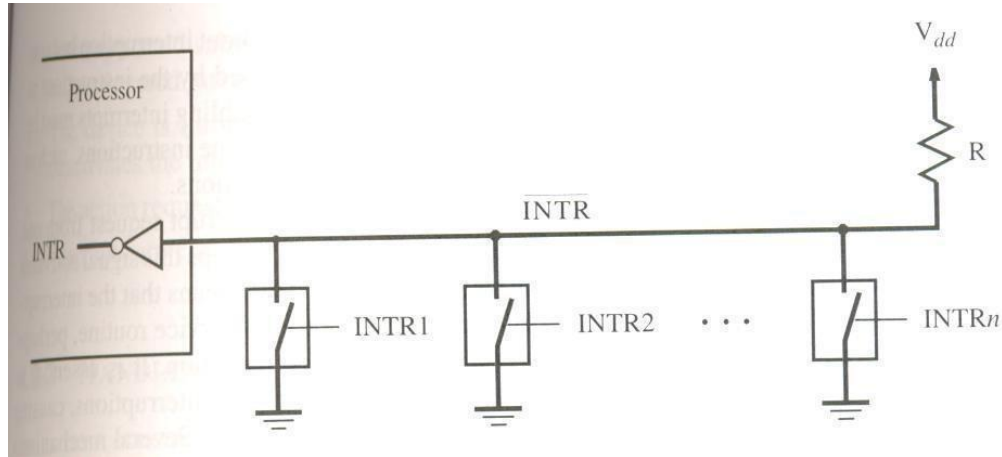
Program 2



- The processor first completes the execution of instruction i . Then it loads the PC(Program Counter) with the address of the first instruction of the ISR.
- After the execution of ISR, the processor has to come back to instruction $i + 1$
- Therefore, when an interrupt occurs, the current contents of PC which point to $i + 1$ is put in temporary storage in a known location.
- A return from interrupt instruction at the end of ISR reloads the PC from that temporary storage location, causing the execution to resume at instruction $i+1$.
- When the processor is handling the interrupts, it must inform the device that its request has been recognized so that it removes its interrupt requests signal.
- This may be accomplished by a special control signal called the **interrupt acknowledge signal**.
- The task of saving and restoring the information can be done automatically by the processor.
- The processor saves only the contents of program counter & status register (ie) it saves only the minimal amount of information to maintain the integrity of the program execution.

- Saving registers also increases the delay between the time an interrupt request is received and the start of the execution of the ISR. This delay is called the **Interrupt Latency**.
- Generally, the long interrupt latency is unacceptable. The concept of interrupts is used in Operating System and in Control Applications, where processing of certain routines must be accurately timed relative to external events. This application is also called as **real-time processing**.

Interrupt Hardware



- A single interrupt request line may be used to serve n devices.
- All devices are connected to the line via switches to ground. To request an interrupt, a device closes its associated switch, the voltage on INTR line drops to 0(zero).
- If all the interrupt request signals (INTR1 to INTRn) are inactive, all switches are open and the voltage on INTR line is equal to Vdd.
- When a device requests an interrupt, the value of INTR is the logical OR of the requests from individual devices. (ie) $INTR = INTR1 + \dots + INTRn$.
- INTR->It is used to name the INTR signal on common line it is active in the low voltage state.
- **Open collector** or **Open drain** is used to drive INTR line.
- The Output of the Open collector (or) Open drain control is equal to a switch to the ground that is open when gates input is in 0 state and closed when the gates input is in 1 state.
- Resistor R is called a **pull-up resistor** because it pulls the line voltage up to the high voltage state when the switches are open.

Enabling and Disabling Interrupts

- The arrival of an interrupt request from an external device causes the processor to suspend the execution of one program & start the execution of another because the interrupt may alter the sequence of events to be executed.
- INTR is active during the execution of **Interrupt Service Routine**.
- There are 3 mechanisms to solve the problem of infinite loop which occurs due to successive interruptions of active INTR signals.
- The following are the typical scenario.

- The device raises an interrupt request.
- The processor interrupts the program currently being executed.
- Interrupts are disabled by changing the control bits in PS (Processor Status register)
- The device is informed that its request has been recognized & in response, it deactivates the INTR signal.
- The actions are enabled & execution of the interrupted program is resumed.

Edge-triggered

- The processor has a special interrupt request line for which the interrupt handling circuit responds only to the leading edge of the signal. Such a line is said to be **edge-triggered**.

Handling Multiple Devices:

- When several devices request an interrupt at the same time, it raises some questions. They are.
 - How can the processor recognize the device requesting an interrupt?
 - Given that the different devices are likely to require different ISRs, how can the processor obtain the starting address of the appropriate routines in each case?
 - Should a device be allowed to interrupt the processor while another interrupt is being serviced?
 - How should two or more simultaneous interrupt requests be handled?

Polling Scheme:

- If two devices have activated the interrupt request line, the ISR for the selected device (first device) will be completed & then the second request can be serviced.
- The simplest way to identify the interrupting device is to have the ISR poll all the encountered with the IRQ bit set is the device to be serviced.
- IRQ (Interrupt Request) -> when a device raises an interrupt request, the status register IRQ is set to 1.

Merit:

- It is easy to implement.

Demerit:

- The time spent for interrogating the IRQ bits of all the devices that may not be requesting any service.

Vectored Interrupt: Nov / Dec 2011, 2012

- Here the device requesting an interrupt may identify itself to the processor by sending a special code over the bus & then the processor starts executing the ISR.
- The code supplied by the processor indicates the starting address of the ISR for the device.
- The code length ranges from 4 to 8 bits. The location pointed to by the interrupting device is used to store the starting address to ISR.
- The processor reads this address, called the interrupt vector & loads into PC.
- The interrupt vector also includes a new value for the Processor Status Register.

- When the processor is ready to receive the interrupt vector code, it activate the interrupt acknowledge (INTA) line.

Interrupt Nesting: Multiple Priority Scheme:

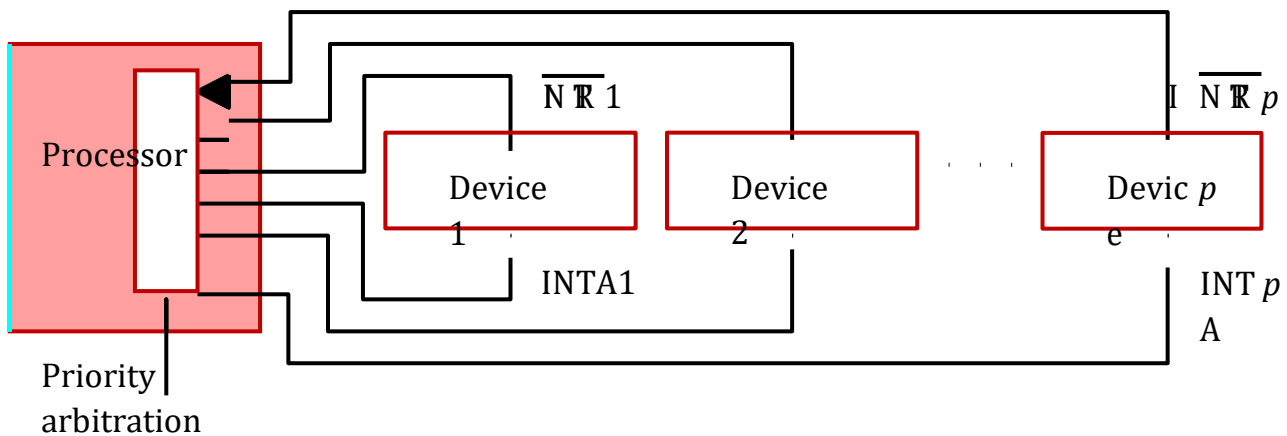
- In multiple level priority schemes, we assign a priority level to the processor that can be changed under program control.
- The priority level of the processor is the priority of the program that is currently being executed.
- The processor accepts interrupts only from devices that have priorities higher than its own.
- At the time the execution of an ISR for some device is started, the priority of the processor is raised to that of the device.
- The action disables interrupts from devices at the same level of priority or lower.

Privileged Instruction:

- The processor priority is usually encoded in a few bits of the Processor Status word.
- It can also be changed by program instruction & then it is writing into PS. These instructions are called **privileged instruction**.
- This can be executed only when the processor is in supervisor mode.
- The processor is in supervisor mode only when executing OS routines. It switches to the user mode before beginning to execute application program.

Privileged Exception:

- User program cannot accidentally or intentionally change the priority of the processor & disrupts the system operation.
- An attempt to execute a privileged instruction while in user mode, leads to a special type of interrupt called the privileged exception.

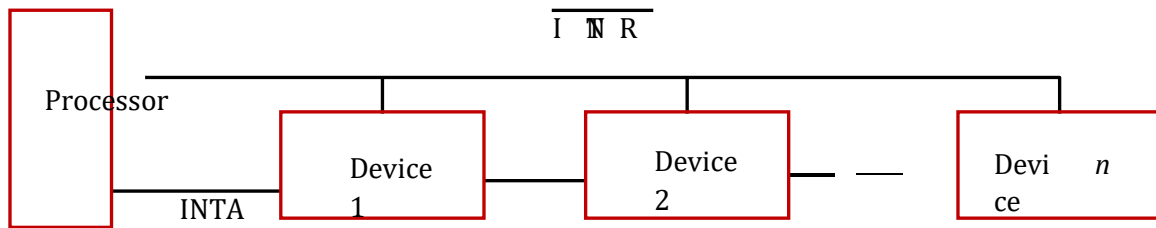


Implementation of Interrupt Priority using individual Interrupt request acknowledge lines

- Each of the interrupt request line is assigned a different priority level.
- Interrupt request received over these lines are sent to a priority arbitration circuit in the processor.
- A request is accepted only if it has a higher priority level than that currently assigned to the processor.

Simultaneous Requests:

Daisy Chain:



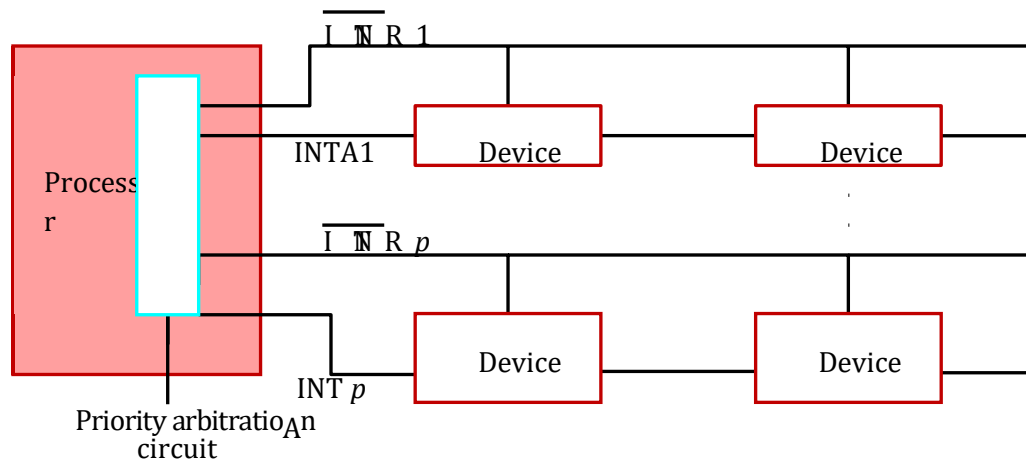
- The interrupt request line INTR is common to all devices.
- The interrupt acknowledge line INTA is connected in a daisy chain fashion such that INTA signal propagates serially through the devices.
- When several devices raise an interrupt request, the INTR is activated & the processor responds by setting INTA line to 1. This signal is received by device.
- Device1 passes the signal on to device2 only if it does not require any service.
- If device1 has a pending request for interrupt blocks that INTA signal & proceeds to put its identification code on the data lines. Therefore, the device that is electrically closest to the processor has the highest priority.

Merits:

- It requires fewer wires than the individual connections.

Arrangement of Priority Groups:

- Here the devices are organized in groups & each group is connected at a different priority level. Within a group, devices are connected in a daisy chain.



- At the devices end, an interrupt enable bit in a control register determines whether the device is allowed to generate an interrupt requests.
- At the processor end, either an interrupt enable bit in the PS (Processor Status) or a priority structure determines whether a given interrupt requests will be accepted.

Initiating the Interrupt Process:

- Load the starting address of ISR in location INTVEC (vectored interrupt).

- Load the address LINE in a memory location PNTR. The ISR will use this location as a pointer to store i/o characters in the memory.
- Enable the keyboard interrupts by setting bit 2 in register CONTROL to 1.
- Enable interrupts in the processor by setting to 1, the IE bit in the processor status register PS.

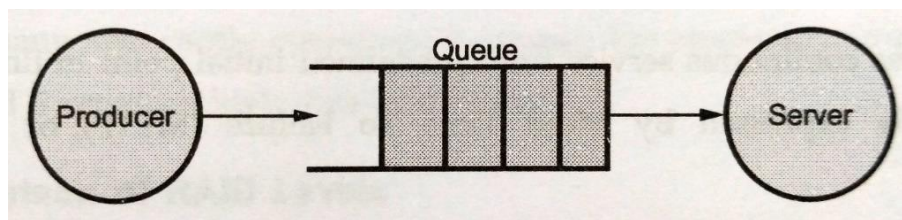
Exception of ISR:

- Read the input characters from the keyboard input data register. This will cause the interface circuits to remove its interrupt requests.
- Store the characters in a memory location pointed to by PNTR & increment PNTR.
- When the end of line is reached, disable keyboard interrupt & inform program main.
- Return from interrupt.

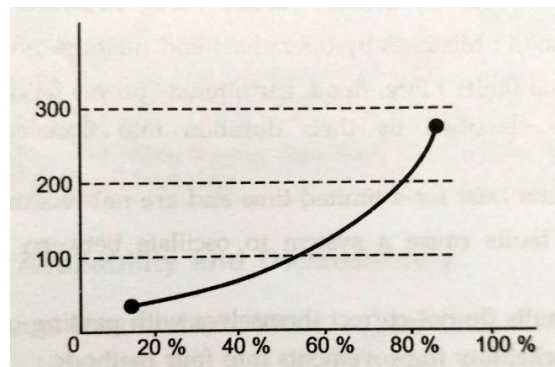
I/O Performance Measures

*13. Explain various ways to List and explain various I/O performance measures. (April/May 2017)
(Or) Explain in detail about I/O performance measures with an example. (April/May 2014,2015)*

- Measures used to quantify I/O performance attempt to measure a more diverse range of properties than the case of CPU performance.
- The traditional measures of performance/ namely response time and throughput/ also apply to I/O. I/O throughput is sometimes called I/O bandwidth/ and response time is sometimes called latency.
- Fig. shows the traditional producer-server model of response time.



- The producer creates tasks to be performed and places them in a buffer; the server takes tasks from the first-in-first-out buffer and performs them.
- Response time and throughput are non-linear. From a transaction server model:
 1. Throughput is maximized when the queue is never empty;
 2. Response time is minimized when die queue is empty.
- Figure below shows throughput versus response time for a typical storage system. Consider two interactive computing environments/ one keyboard driven/ one graphical.



- Computing interaction or transaction time is divided into three components,
 1. **Entry Time** : Time for user to make a request;
 2. **System Response Time** : Time between request and response;
 3. **Think Time** : Time between system response and next request
- The sum of these three parts is called the transaction time. Several studies report that user productivity is inversely proportional to transaction time; transactions per hour are a measure of the work completed per hour by the user.
- System response time is naturally the shortest duration. Does this minimize the impact of response time?
- Effect of system response time on user 'thinking' time.
 1. Any reduction to response time has more than a linear reduction on total transaction time.
 2. Users need less time to think when given a faster response;
 3. Possible to attach an economic benefit to response time and throughput;
 4. In order to maintain user interest, response times need to be < 1.0 second.

Problem: [May 2019]

Suppose a processor sends 80 disks I/Os per second, these requests are exponentially distributed, and the average service time of an older disk is 25 ms.

Answer the following questions:

1. **On average, how utilized is the disk?**
2. **What is the average time spent in the queue?**
3. **What is the average response time for a disk request, including the queuing time and disk service time?**

Average number of Arriving tasks / second is 80 ms.

Average disk time to service a task is 25ms = (0 .025 sec).

The server utilization is then

Service utilization = Arrival rate x Time_{Server}

$$= 80 \times 0.025 = 2.0$$

Since the service times are exponentially distributed, we can use the simplified formula for the average time spent waiting in line:

$$Time_{Queue} = Time_{Server} \times \frac{server\ Utilization}{(1 - Server\ Utilization)}$$

$$= 25\ ms \times \frac{2}{1 - 2}$$

$$= 50\ ms\ (\text{consider +ve value})$$

The average response time is

Time system = Time_{queue} + Time_{Server}

$$= 50\ ms + 25\ ms$$

= 75 ms

Thus, on average we spend 75% of our time waiting in the queue!

Universal Serial Bus (USB)

14. Explain about USB.

- The **universal serial bus (USB)** is a standard interface for connecting a wide range of devices to the computer such as keyboard, mouse, smart phones, speakers, cameras etc.
- The USB was introduced for commercial use in the year 1995 at that time it has a data transfer speed of 12 megabits/s.
- With some improvement, a modified USB 2 was introduced which is also called a high speed USB that transfers data at 480 megabits/s.
- With the evolution of I/O devices that require high speed data transfer also leads to the development of USB 3 which is also referred to as Super speed USB which transfers data at 5 gigabits/s.
- The recent version of USB can transfer data up to 20 gigabits/s.

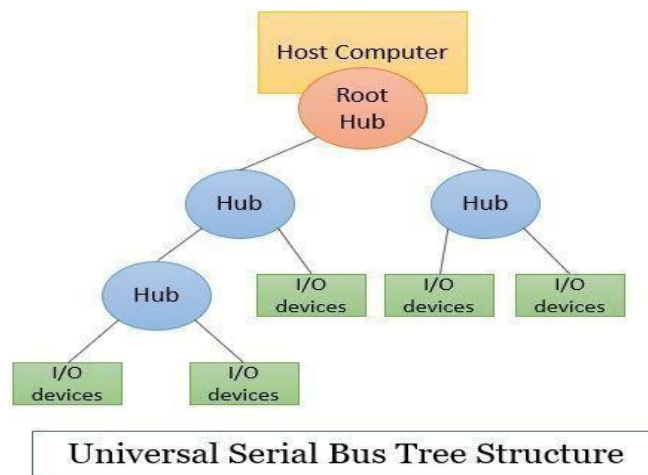
Content: Universal Serial Bus (USB)

1. Key Objectives
2. USB Architecture
3. Isochronous Traffic on USB
4. Types of USB Connectors
5. Electrical Characteristics of USB

Key Objectives of Universal Serial Bus

- Before getting into the details of the universal serial bus we will discuss some of the key objectives that are taken into account while designing a USB.
- The developed USB must be simple and a low-cost interconnection system that should be easy to use.
- The developed USB must be compatible with all new I/O devices, their bit rates, internet connections and audio, video application.
- The USB must support a plug-and-play mode of operation.
- The USB must support low power implementation.
- The USB must also provide support for legacy hardware and software.

USB Architecture



- When multiple I/O devices are connected to the computer through USB they all are organized in a tree structure.
- Each I/O device makes a point-to-point connection and transfers data using the serial transmission format we have discussed serial transmission in our previous content `__interface circuit`.
- As we know a tree structure has a **root**, **nodes** and **leaves**.
- The tree structure connecting I/O devices to the computer using USB has nodes which are also referred to as a **hub**.
- Hub is the inter-mediatory connecting point between the I/O devices and the computer.
- Every tree has a root here; it is referred to as the **root hub** which connects the entire tree to the hosting computer.
- The leaves of the tree here are nothing but the I/O devices such as a mouse, keyboard, camera, and speaker.
- The USB works on the principle of polling.
- In **polling**, the processor keeps on checking whether the I/O device is ready for data transfer or not.
- So, the devices do not have to inform the processor about any of their statuses.
- It is the processor's responsibility to keep a check. This makes the USB simple and low cost.
- Whenever a new device is connected to the hub it is addressed as 0.
- Now at a regular interval the host computer polls all the hubs to get their status which lets the host know of I/O devices that are either detached from the system or are attached to the system.
- When the host becomes aware of the new device it gets to know about the capabilities of the device by reading the information present in the special memory of the device's USB interface.
- So that the host can use the appropriate device driver to communicate with the device.
- The host then assigns an address to this new device, this address is written to the register of the device interface register.
- With this mechanism, USB serves plug-and-play capability.

- The **plug and play** feature let the host recognize the existence of the new I/O device automatically when the device is plugged in.
- The host software determines the capabilities of the I/O devices and if it has any special requirement.
- The USB is **hot-pluggable** which means the I/O device can be attached or removed from the host system without performing any restart or shutdown.
- That means your system can keep running while the I/O device is plugged or removed.

Isochronous Traffic on USB

- USB also supports the isochronous traffic where the data is transferred at a fixed timed interval, where the intervals are regular and of very short time.
- The isochronous data transmission is comparatively faster than asynchronous and synchronous data transfer.
- To accommodate the isochronous traffic, the root hub sends a sequence of bits over the USB tree this indicates the start of isochronous data and after this sequence of bits, the actual data is transmitted.
- As USB support the isochronous data transmission the audio-video signals are transferred in a precisely timely manner.

Types of USB Connectors

- The USB has different types of ports and connectors.
- Usually, the upstream port and connector are always the USB type A the downstream port and connector differ depending on the type of device connected.

USB Type A:

- This is the standard connector that can be found at one end of the USB cable and is also known as upstream.
- It has a flat structure and has four connecting lines as you can see in the image below.

USB Type B:

- This is an older standard cable and was used to connect the peripheral devices also referred to as downstream.
- It is approximately a square as you can see in the image below.
- This is now been replaced by the newer versions.

Mini USB:

- This type of USB is compatible with mobile devices.
- This type of USB is now superseded your micro-USB still you will get it on some devices.

Micro USB:

- This type of USB is found on newer mobile devices. It has a compact 5 pin design.

USB Type C:

- This type of USB is used for transferring both data and power to the attached peripheral or I/O device.
- The USB C does not have a fixed orientation as it is reversible i.e. you can plug it upside down or in reverse.

USB 3.0 Micro B:

- This USB is a super speed USB. This USB is used for a device that requires high-speed data transfer.
- You can find this kind of USB on portable hard drives.

Electrical Characteristics of USB

- The standard USB has four lines of connection among which two carry power (one carry +5 V and one is for Ground).
- The other two lines of connection are for data transfer.
- USB also supply power to connected I/O device that requires very low power.
- Transferring of data over USB can be divided into two categories i.e., transferring data at low speed and transferring data at high speed.
- The low-speed transmission uses **single-ended signaling** where varying high voltage is transmitted over one of the two data lines to represent the signal bit 0 or 1.
- The other data line is connected to the reference voltage i.e., ground.
- The single-ended signaling is prone to noise.
- The high-speed data transmission uses the approach **differential signaling**.
- Here, the signal is transmitted over the two data lines that are twisted together.
- Here both the data lines are involved in carrying the signal no ground wire is required.
- The differential signaling is not prone to noise and uses low voltages as compared to single-ended transmission.
- So, this is all about the universal serial bus which connects the I/O devices to the host computer. We have seen how it works and how many versions of USB we have.

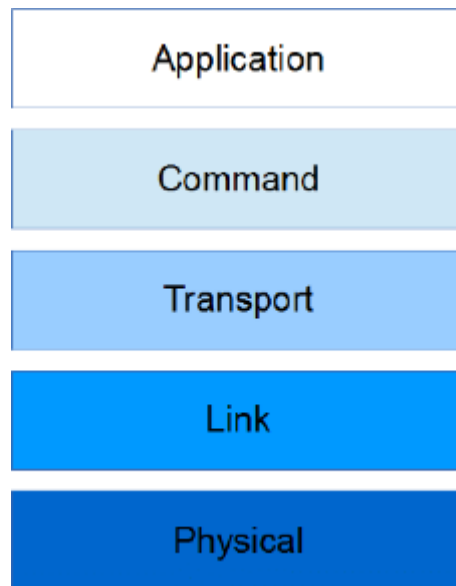
SATA

15. Explain about SATA.

- Serial ATA is a peripheral interface created in 2003 to replace Parallel ATA, also known as IDE.

- Hard drive speeds were getting faster, and would soon outpace the capabilities of the older standard—the fastest PATA speed achieved was 133MB/s, while SATA began at 150MB/s and was designed with future performance in mind.
- Also, newer silicon technologies used lower voltages than PATA's 5V minimum.
- The ribbon cables used for PATA were also a problem; they were wide and blocked air flow, had a short maximum length restriction, and required many pins and signal lines.
- SATA has a number of features that make it superior to Parallel ATA. The signaling voltages are low and the cables and connectors are very small.
- SATA has outpaced hard drive performance, so the interface is not a bottleneck in a system.
- It also has a number of new features, including hot-plug support. SATA is a point-to-point architecture, where each SATA link contains only two devices: a SATA host (typically a computer) and the storage device.
- If a system requires multiple storage devices, each SATA link is maintained separately. This simplifies the protocol and allows each storage device to utilize the full capabilities of the bus simultaneously, unlike in the PATA architecture where the bus is shared.
- To ease the transition to the new standard, SATA maintains backward compatibility with PATA.
- To do this, the Host Bus Adapter (HBA) maintains a set of shadow registers that mimic the registers used by PATA. The disk also maintains a set of these registers.
- When a register value is changed, the register set is sent across the serial line to keep both sets of registers synchronized.
- This allows for the software drivers to be agnostic about the interface being used.
- Serial ATA is a peripheral interface created in 2003 to replace Parallel ATA, also known as IDE.
- Hard drive speeds were getting faster, and would soon outpace the capabilities of the older standard—the fastest PATA speed achieved was 133MB/s, while SATA began at 150MB/s and was designed with future performance in mind.
- Also, newer silicon technologies used lower voltages than PATA's 5V minimum. The ribbon cables used for PATA were also a problem; they were wide and blocked air flow, had a short maximum length restriction, and required many pins and signal lines.
- SATA has a number of features that make it superior to Parallel ATA.
- The signaling voltages are low and the cables and connectors are very small. SATA has outpaced hard drive performance, so the interface is not a bottleneck in a system. It also has a number of new features, including hot-plug support.
- SATA is a point-to-point architecture, where each SATA link contains only two devices: a SATA host (typically a computer) and the storage device.
- If a system requires multiple storage devices, each SATA link is maintained separately. This simplifies the protocol and allows each storage device to utilize the full capabilities of the bus simultaneously, unlike in the PATA architecture where the bus is shared.

- To ease the transition to the new standard, SATA maintains backward compatibility with PATA.
- To do this, the Host Bus Adapter (HBA) maintains a set of shadow registers that mimic the registers used by PATA.
- The disk also maintains a set of these registers. When a register value is changed, the register set is sent across the serial line to keep both sets of registers synchronized.
- This allows for the software drivers to be agnostic about the interface being used.



Physical Layer:

- The physical layer is the lowest layer of the SATA protocol stack. It handles the electrical signal being sent across the cable.
- The physical layer also handles some other important aspects, such as resets and speed negotiation.
- SATA uses low-voltage differential signaling (LVDS). Instead of sending 1's and 0's relative to a common ground, the data being sent is based on the difference in voltage between two conductors sending data.
- In other words, there is a TX+ and a TX- signal. A logic 1 corresponds to a high TX+ and a low TX-; and vice versa for a logic 0. SATA uses a $\pm 125\text{mV}$ voltage swing.

Link Layer

- The link layer is the next layer and is directly above the physical layer. This layer is responsible for encapsulating data payloads and manages the protocol for sending and receiving them.
- A data payload that is sent is called a Frame Information Structure (FIS). The link layer also provides some other services for ensuring data integrity, handling flow control, and reducing EMI.
- The host and the disk each have their own transmit pair in a SATA cable, and theoretically data could be sent in both directions simultaneously.
- However, this does not occur. Instead, the receiver sends -backchannel information to the sender that indicates the status of the transfer in progress.

- For instance, if an error were to be detected mid- transmission, such as a disparity error, the receiver could notify the sender of this.

Transport Layer

- The transport layer is responsible for constructing, delivering, and receiving Frame Information Structures.
- It defines the format of each FIS and the valid sequence of FISes that can be exchanged.
- The first byte of each FIS defines the type. The second byte contains type- dependent control fields.
- The following table lists some of the types of FISes that are defined, and the value of their type field.

FIS Type	Type Value
Register - Host to Device	0x27
Register - Device to Host	0x34
Data	0x46
DMA Activate	0x39

Table 4 FIS Types

2 Marks
Question Bank

1. What is Memory?

- Memory is a device used to store the data and instructions required for any operation.

2. What is the secondary memory?

- Secondary memory is where programs and data are kept on a long-term basis. Common secondary storage devices are the hard disk and optical disks. The hard disk has enormous storage capacity compared to main memory. The hard disk is usually contained inside the case of a computer.

3. What are some examples of secondary storage device?

- Some other examples of secondary storage technologies are flash memory (e.g. USB flash drives or keys), floppy disks, magnetic tape, paper tape, punched cards, standalone RAM disks, and Iomega Zip drives.

4. What are the characteristics of a secondary storage device?

- Characteristics of a secondary storage devices are,
- Capacity
- Speed
- Portability
- Durability
- Reliability

5. What are the three main categories of secondary storage?

Currently the most common forms of secondary storage device are:

- Floppy disks
- Hard disks
- Optical Disks
- Magnetic Tapes
- Solid State Devices

6. What is Bandwidth?

- The maximum amount of information that can be transferred to or from the memory per unit time is called bandwidth.

7. Define a Cache.

- It is a small fast intermediate memory between the processor and the main memory.

8. What is Cache Memory?

- Cache memory is a very high speed memory that is placed between the CPU and primary or main memory.

- It is used to reduce the average time to access data from the main memory.
- The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations.
- Most CPUs have different independent caches, including instruction and data.

9. Give the mapping techniques of cache.

The three different types of mapping techniques used for the purpose of cache memory are as follow,

- ✓ Direct Mapping
- ✓ Associative Mapping
- ✓ Set-Associative Mapping

10. What is Write Stall?

- When the processor must wait for writes to complete during write through, the processor caches is said to write stall.

11. Define Mapping Functions.

- The correspondence of memory blocks in cache with the memory blocks in the main memory is defined as mapping functions.

12. What is Address Translation?

- The conversion of virtual address to physical address is termed as address translation.

13. What is the transfer time?

- The time it takes to transmit or move data from one place to another.
- It is the time interval between starting the transfer and the completion of the transfer.

(Or)

- Transfer time is the time it takes to transfer a block of bits, typically a sector under the read / write head.

14. What is latency and seek time?

- **Seek Time** is measured defines the amount of time it takes a hard drive's read/write head to find the physical location of a piece of data on the disk.
- **Latency** is the average time for the sector being accessed to rotate into position under a head, after a completed seek.

15. What is the clock cycle time?

- The speed of a computer processor, or CPU, is determined by the clock cycle, which is the amount of time between two pulses of an oscillator.
- Computer processors can execute one or more instructions per clock cycle, depending on the type of processor.

16. What is Access Time?

- The time a program or device takes to locate a single piece of information and make it available to the computer for processing. *DRAM (dynamic random access memory)* chips for personal computers have access times of 50 to 150 nanoseconds (billionths of a second).

17. What is meant by disk fragmentation?

- Fragmentation refers to the condition of a disk in which files are divided into pieces scattered around the disk.
- It occurs naturally when you use a disk frequently, creating, deleting, and modifying files.
- At some point, the operating system needs to store parts of a file in noncontiguous clusters.

18. What is the average access time for a hard disk?

- Disk access times are measured in milliseconds (thousandths of a second), often abbreviated as ms.
- Fast hard disk drives for personal computers boast access times of about 9 to 15 milliseconds.
- Note that this is about 200 times slower than average DRAM.

19. What is rotational latency time?

- The amount of time it takes for the desired sector of a disk (i.e., the sector from which data is to be read or written) to rotate under the read-write heads of the disk drive.
- It is also called as rotational delay.

20. What is meant by disk latency?

- Disk latency refers to the time delay between a request for data and the return of the data. It sounds like a simple thing, but this time can be critical to the performance of a system.

21. Define Page Fault.

- If the processor access for the particular page in main memory and if the page is not present there then it is known as page fault.

22. Define a Cache Unit.

- When the CPU refers to memory and finds a required word in cache it is termed as cache hit.

23. Define Hit Ratio.(Nov/Dec 2019)Nov/Dec 2021

- The ratio of the number of hits divided by the total CPU references to memory is the hit ratio.

24. Define a Miss.Nov/Dec 2021

- When the CPU refers to memory and if the required word is not found in cache it is termed as miss.

25. What is meant by memory stall cycles? (M 2016)

- The number of cycles during which the CPU is stalled waiting for a memory access is called memory stall cycles.

26. What is Miss Penalty?

- The number of stall cycles depends on both the number of misses and the cost per miss, which is called the miss penalty.

27. Write the formula to calculate average memory access time. (Or) Write the formula to measure average memory access time for memory hierarchy performance. (Nov / Dec 2018)

- Average memory access time = Hit time + Miss rate x Miss penalty

28. What is a miss in a cache? (Or) What does Cache Miss mean? (Or) Define Cache Miss. (Nov/Dec 2010, April/May 2018)

- Cache miss is a state where the data requested for processing by a component or application is not found in the cache memory.
- It causes execution delays by requiring the program or application to fetch the data from other cache levels or the main memory.

29. Define Cache Hit. (Or) What does Cache Hit mean? (April/May 2018)

- A cache hit is a state in which data requested for processing by a component or application is found in the cache memory.
- It is a faster means of delivering data to the processor, as the cache already contains the requested data.

30. Differentiate between Cache Miss and Cache Hit.

- The difference between the two is the data requested by a component or application in a cache memory being found or not.
- In a cache miss, the data is not found so the execution is delayed because the component or application tries to fetch the data from main memory or other cache levels.
- In a cache hit, the data is found in the cache memory making it faster to process.
- The **cache hit** is when you look something up in a cache and it *was* storing the item and is able to satisfy the query.

31. What is miss penalty for a cache?

- Cache is a small high-speed memory. Stores data from some frequently used addresses (of main memory). Processor loads data from M and copies into cache.
- This results in extra delay, called miss penalty.
- Hit ratio = percentage of memory accesses satisfied by the cache.

32. What is miss rate in cache?

- The fraction or percentage of accesses that result in a hit is called the hit rate.
- The fraction or percentage of accesses that result in a miss is called the miss rate.
- It follows that hit rate + miss rate = 1.0 (100%).
- The difference between lower level access time and cache access time is called the miss penalty.

33. What is hit time in cache?

- AMAT's three parameters hit time (or hit latency), miss rate, and miss penalty provide a quick analysis of memory systems.

- Hit latency (H) is the time to hit in the cache. Miss rate (MR) is the frequency of cache misses, while average miss penalty (AMP) is the cost of a cache miss in terms of time.

34. How is cache memory measured?

- The CPU cache is a piece of hardware which reduces the access time to the data in the memory by keeping some part of the frequently used data of the main memory in itself.
- It is smaller and faster than the main memory.

35. What is the memory cycle time?

- Cycle time is the time, usually measured in nanoseconds, between the start of one random access memory (RAM) access to the time when the next access can be started.
- Access time is sometimes used as a synonym (although IBM deprecates it).

36. What is the memory access time?

- Memory access time is how long it takes for a character in memory to be transferred to or from the CPU.
- In a PC or Mac, fast RAM chips have an access time of 70 nanoseconds (ns) or less.
- SDRAM chips have a burst mode that obtains the second and subsequent characters in 10 ns or less.

37. What is the data transfer rate?

- The speed with which data can be transmitted from one device to another.
- Data rates are often measured in megabits (million bits) or megabytes (million bytes) per second.
- These are usually abbreviated as Mbps and MBps, respectively. Another term for data transfer rate is throughput.

38. What does access time measure?

- The total time it takes the computer to read data from a storage device such as computer memory, hard drive, CD-ROM or other mechanism.
- Computer access time is commonly measured in nanoseconds or milliseconds and the lower the access time the better.

39. List the method to improve the cache performance.

Improving the cache performance following methods are used:

- Reduce the miss rate.
- Reduce the miss penalty.
- Reduce the time to hit in the cache.

40. What is Split Transactions?

- With multiple masters, a bus can offer higher bandwidth by using packets, as opposed to holding the bus for the full transaction. This technique is called split transactions.

41. What is Cylinder?

- Cylinder is used to refer to all the tracks under the arms at a given points on all surfaces.

42. What is Synchronous Bus?

- Synchronous bus includes a clock in the control lines and a fixed protocol for sending address and data relative to the clock.

43. Explain difference between latency and throughput.

- Latency is defined as the time required processing a single instruction, while throughput is defined as the number of instructions processes per second.

44. What is called Pages?

- The address space is usually broken into fixed-size blocks, called pages. Each page resides either in main memory or on disk.

45. What are the techniques to reduce hit time?

The techniques to reduce hit time are:

- Small and simple cache: Direct mapped.
- Avoid address translation during indexing of the cache.
- Pipelined cache access.
- Trace cache.

46. What are the categories of cache miss? (April/May 2013) (Or) Point out one simple technique used to reduce each of the three "C" misses in cache memories. (Nov/Dec 2017)

- Categories of cache misses are,
 - ✓ Compulsory
 - ✓ Capacity
 - ✓ Conflict

47. How the conflicts misses are divided? (Nov/Dec 2016)

Four divisions of conflict misses are:

- **Eight way:** Conflict misses due to going from fully associative to eight way associative.
- **Four way:** Conflict misses due to going from eight way associative to four way associative.
- **Two way:** Conflict misses due to going from four way associative to two way associative.
- **One way:** Conflict misses due to going from two way associative to one way associative.

48. What is Sequence Recorded?

- The sequence recorded on the magnetic medias is a sector number, a gap, the information for that sector including error correction cede, a gap, the sector number of the next sector and so on.

49. Write the formula to calculate the CPU execution time.

- CPU execution time = (CPU clock cycles + Memory stall cycles) x Clock cycle time.

50. Write the formula to calculate the CPU time.

- CPU time = (CPU execution clock cycles + Memory stall clock cycles) x Clock cycle time.

51. What is RAID? (Nov/Dec 2011, April/May 2015, 2017)

- RAID stands for Redundant Array of Independent Disks.
- It is also called as redundant array of inexpensive disks.

- It is a way of storing the same data in different places on multiple hard disks.

(Or)

- **RAID** is a storage technology that combines multiple disk drive components into a logical unit for the purposes of data redundancy and performance improvement.
- Data is distributed across the drives in one of several ways, referred to as RAID levels, depending on the specific level of redundancy and performance required.

52. Explain the terms availability and dependability. (Nov/Dec 2017)

- Availability is a measure of the service accomplishment with respect to the alternation between the two states of accomplishment and interruption.
- Dependability is the quality of delivered service such that reliance can justifiably be placed on this service.

53. What are the differences and similarities between SCSI and IDE? (April/May 2017)

Parameters	IDE	SCSI
Cost	IDE is a much cheaper solution	SCSI is often more expensive to implement and support
Expansion	It allows 2 two devices per channel.	It is capable of supporting up to 7 or 15 devices.
Ease	IDE is commonly a much easier product to setup than SCSI.	Configuring SCSI can be more difficult for most users when compared to IDE.
CPU	IDE devices cannot communicate independently from the CPU.	SCSI devices can communicate independently from the CPU over the SCSI bus.

54. Why does DRAM generally have much larger capacities than SRAMs constructed in the same fabrication technology? (Nov/Dec 2016)

- DRAM bit cells require only two devices (capacitor and transistor) while SRAM bit cells typically requires six transistors.
- This makes the bit cells of the DRAM much smaller than the bit cells of the SRAM, allowing the DRAM to store more data in the same amount of chip space.

55. What are the measures of I/O performance? (Nov/Dec 2013)

I/O Performance Measures

- Measures used to quantify I/O performance attempt to measure a more diverse range of properties than the case of CPU performance.
- The traditional measures of performance namely response time and throughput also apply to I/O. I/O throughput is sometimes called I/O bandwidth and response time is sometimes called latency.

56. What are the types of storage devices? (Nov/Dec 2016, April/May 2017)

- Physical components or materials on which data is stored are called storage media.
- Hardware components that read/write to storage media are called storage devices. A floppy disk drive is a storage device.
- Two main categories of storage technology used today are magnetic storage and optical storage. Storage devices hold data, even when the computer is turned off. The physical material that actually holds data is called storage medium.

57. What do you mean by Memory Interleaving?

- Interleaved memory is a design made to compensate for the relatively slow speed of dynamic random-access memory (DRAM) or core memory, by spreading memory addresses evenly across memory banks.

58. What are the factors responsible for the maximum I/O bus performance? (April/May 2005)

Factors to be considered for maximum I/O bus performance are:

- Latency & Bandwidth

59. What are the two major advantages and disadvantages of the bus? (May/June 2007)

Advantages:

- It is easy to set-up and extend bus network.
- Cable length required for this topology is the least compared to other networks. Bus topology costs very less.
- Bus topology costs very less.

Disadvantages:

- There is a limit on central cable length and number of nodes that can be connected
- Dependency on central cable in this topology has its disadvantages. If the main cable (i.e. bus) encounters some problem, whole network breaks down.

60. Is the RISC processor is necessary? Why? (May/June 2007)

- Although RISC was indeed able to scale up in performance quite quickly and cheaply and it is necessary for building block of high performance parallel processors.

61. Define the terms cache miss and cache hit. (Nov/Dec 2011, May/June 2013)

- A *cache miss*, generally, is when something is looked up in the cache and is not found.
- The cache did not contain the item being looked up. The *cache hit* is when you look something up in a cache and it *was* storing the item and is able to satisfy the query.

62. Compare software and hardware RAID.

Software RAID	Hardware RAID
1. A simple way to describe software RAID is that the RAID task runs on the CPU of your computer system. 2. It is implemented in Pure Software model –	1. A hardware RAID solution has its own processor and memory to run the RAID application. In this implementation, the RAID system is an independent small computer system

Operating System Software and hybrid model-Assisted Software RAID.

dedicated to the RAID application, offloading this task from the host system.

2. Hardware RAID can be implemented in a variety of ways:
 - as a discrete RAID Controller Card, or
 - as integrated hardware based on RAID-on-Chip technology

63. Explain the need to implement memory as a hierarchy. (April/May 2017)

- A "memory hierarchy" in computer storage distinguishes each level in the "hierarchy" by response time.
- Each level of the hierarchy is of higher speed and lower latency, and is of smaller size, than lower levels.
- The levels in the memory hierarchy not only differ in speed and cost. They are performing different roles.

64. What is a register in memory?

- A register is a very small amount of very fast memory that is built into the CPU (central processing unit) in order to speed up its operations by providing quick access to commonly used values.
- Registers are the top of the memory hierarchy and are the fastest way for the system to manipulate data.

65. What is the storage hierarchy?

- A storage device hierarchy consists of a group of storage devices that have different costs for storing data, different amounts of data stored, and different speeds of accessing the data.
- Level 0, including DFSMSHsm-managed storage devices at the highest level of the hierarchy, contains data directly accessible to you.

66. What is Cache Optimization?

- The idea behind this approach is to hide both the low main memory bandwidth and the latency of main memory accesses which is slow in contrast to the floating-point performance of the CPUs.

67. List the six basic optimization techniques of cache. (Nov/Dec 2016) (Or) List the basic six cache optimizations for improving cache performance. (Nov / Dec 2018)

- The six basic optimization techniques of cache are,
 - ✓ Larger block size to reduce miss rate
 - ✓ Bigger caches to reduce miss rate
 - ✓ Higher associativity to reduce miss rate
 - ✓ Multilevel caches to reduce miss penalty
 - ✓ Giving priority to read misses over writes to reduce miss penalty
 - ✓ Avoiding address translation during indexing of the cache to reduce hit time

68. Difference between Volatile and Non-volatile Memory. (Or) Outline the difference between volatile and non-volatile memory. (April/May 2018)

Volatile Memory (RAM)	Non-volatile Memory (ROM)
<ul style="list-style-type: none"> → It is a volatile memory. → Contents are stored temporarily. → Cost is very high. → Small storage capacity. → Processing speed is high. 	<ul style="list-style-type: none"> → It is a non-volatile memory. → Contents are stored permanently. → Cost Effective. → High storage capacity. → Processing speed is low.

69. What is a Cache Performance?

- Cache Performance - Average memory access time is a useful measure to evaluate the performance of a memory-hierarchy configuration.

70. What is hit and miss ratio?

- The hit ratio is the fraction of accesses which are a hit.
- The miss ratio is the fraction of accesses which are a miss. It holds that. miss rate = 1 – hit rate.
- The (hit/miss) latency (AKA access time) is the time it takes to fetch the data in case of a hit/miss.

71. Differentiate between SRAM and DRAM.

- SRAM (Static RAM) and DRAM (Dynamic RAM) holds data but in a different ways.
- DRAM requires the data to be refreshed periodically in order to retain the data.
- SRAM does not need to be refreshed as the transistors inside would continue to hold the data as long as the power supply is not cut off.
- DRAM memory slower and less desirable than SRAM.

72. Differentiate between throughput and response time. [May 2019]

Throughput Time	Response Time
<ul style="list-style-type: none"> 1. This is the time difference between submission of a request until the response begins to be received. 2. The response time should be as low as possible so that a large number of interactive users receive an acceptable response time. 	<ul style="list-style-type: none"> 1. The number of processes that are completed per unit time is called the throughput. 2. It is desirable to maximize CPU utilization and throughput and to minimize turnaround time and response time.

73. Suppose a processor sends 80 disk I/Os per second, these requests are exponentially distributed, and the average service time of an older disk is 25 ms. Nov/Dec 2020.

Answer the following questions:

- 1. On average, how utilized is the disk?**
- 2. What is the average time spent in the queue?**
- 3. What is the average response time for a disk request, including the queuing time and disk service time?**

Average number of Arriving tasks / second is 80 ms.

Average disk time to service a task is 25ms = (0 .025 sec).

The server utilization is then

$$\begin{aligned}\text{Service utilization} &= \text{Arrival rate} \times \text{Time}_{\text{Server}} \\ &= 80 \times 0.025 = 2.0\end{aligned}$$

Since the service times are exponentially distributed, we can use the simplified formula for the average time spent waiting in line:

$$\begin{aligned}\text{Time}_{\text{Queue}} &= \text{Time}_{\text{Server}} \times \frac{\text{server Utilization}}{(1 - \text{Server Utilization})} \\ &= 25 \text{ ms} \times \frac{2}{1 - 2} \\ &= 50 \text{ ms (consider +ve value)}\end{aligned}$$

The average response time is

$$\begin{aligned}\text{Time system} &= \text{Time}_{\text{queue}} + \text{Time}_{\text{Server}} \\ &= 50 \text{ ms} + 25 \text{ ms} \\ &= 75 \text{ ms}\end{aligned}$$

Thus, on average we spend 75% of our time waiting in the queue!

74. What is IO mapped input output?

- A memory reference instruction activated the READ M (or) WRITE M control line and does not affect the IO device. Separate IO instruction is required to activate the READ IO and WRITE IO lines, which cause a word to be transferred between the address port and the CPU.
- The memory and IO address space are kept separate.

75. Specify the three types of the DMA transfer techniques?

- Single transfer mode (cyclestealing mode)
- Block Transfer Mode (Burst Mode)
- Demand Transfer Mode
- Cascade Mode

76. Name any three of the standard I/O interface.

- SCSI (small computer system interface), bus standards
- Back plane bus standards
- IEEE 796 bus (multibus signals)
- NUBUS & IEEE 488 bus standard

77. What is an I/O channel?

- An I/O channel is actually a special purpose processor; also called peripheral processor.
- The main processor initiates a transfer by passing the required information in the input output channel. The channel then takes over and controls the actual transfer of data.

78. Why program controlled I/O is unsuitable for high-speed data transfer?

- In program controlled I/O considerable overhead is incurred. Because several program instructions have to be executed for each data word transferred between the external devices and MM.
- Many high speed peripheral; devices have a synchronous modes of operation. That is data transfer is controlled by a clock of fixed frequency, independent of the CPU.

79. What is the function of I/O interface?

- The function is to coordinate the transfer of data between the CPU and external devices.

80. Name some of the IO devices.

- Video terminals
- Video displays
- Alphanumeric displays
- Graphics displays
- Flat panel displays
- Printers
- Plotters

81. Define interface.

- The word interface refers to the boundary between two circuits or devices

82. What is programmed I/O?

- Data transfer to and from peripherals may be handled using this mode. Programmed I/O operations are the result of I/O instructions written in the computer program.

83. What are the limitations of programmed I/O? (Nov / Dec 2011)

The main limitation of programmed I/O and interrupt driven I/O is given below:

Programmed I/O

- It used only in some low-end microcomputers.
- It has single input and single output instruction.
- Each instructions selects one I/O device (by number) and transfers a single character (byte)
- Example: microprocessor controlled video terminal.
- Four registers: input status and character, output status and character.

Interrupt-driven I/O

- Primary disadvantage of programmed I/O is that CPU spends most of its time in a tight loop waiting for the device to become ready. This is called busy waiting.
- With interrupt-driven I/O, the CPU starts the device and tells it to generate an interrupt when it is finished.
- Done by setting interrupt-enable bit in status register.
- Still requires an interrupt for every character read or written.
- Interrupting a running process is an expensive business (requires saving context).

- Requires extra hardware (DMA controller chip).

84. Differentiate Programmed I/O and Interrupt I/O. (Nov / Dec 2014)

Programmed I/O	Interrupt I/O
In programmed I/O, processor has to check each I/O device in sequence and in effect ‘ask’ each one if it needs communication with the processor. This checking is achieved by continuous polling cycle and hence processor cannot execute other instructions in sequence.	External asynchronous input is used to tell the processor that I/O device needs its service and hence processor does not have to check whether I/O device needs its service or not.
During polling processor is busy and therefore, has serious and decremental effect on system throughput.	In Interrupt driven I/O, the processor is allowed to execute its instructions in sequence and only stop to service I/O device when it is told to do so by the device itself. This increases system throughput.
It is implemented without interrupt hardware support.	It is implemented using interrupt hardware support.
It does not depend on interrupt status.	Interrupt must be enabled to process Interrupt driven I/O.
It does not need initialization of stack.	It needs initialization of stack.
System throughput decreases as number of I/O devices connected in the system increases.	System throughput does not depend on number of I/O devices connected in the system.

85. Differentiate between memory-mapped I/O and I/O mapped I/O. (Apr / May 2011)

S.No.	Parameter	Memory-mapped I/O	I/O-mapped I/O
1.	Address space	Memory and I/O devices share the entire address space	Memory and I/O devices have separate address space
2.	Hardware	No additional hardware required	Additional hardware required
3.	Implementation	Easy to implement	Difficult to implement
4.	Address	Same address cannot be used to refer both memory and I/O device.	Same address can be used to refer both memory and I/O device.
5.	Control lines	Memory control lines are used to control I/O devices.	Different set of control lines are used to control memory and I/O.
6.	Control lines used	The control lines are: READ, WRITE	The control lines are: READ M, WRITE M, READ I/O, WRITE I/O

86. What is DMA?

- A special control unit may be provided to enable transfer a block of data directly between an external device and memory without contiguous intervention by the CPU. This approach is called DMA.
- (OR)

- A direct memory access (DMA) is an operation in which data is copied (transported) from one resource to another resource in a computer system without the involvement of the CPU.

87. What are MAR and MBR?

- MAR – Memory address register holds the address of the data to be transferred.
- MBR – Memory buffer register contains the data to be transferred to or from the main memory.

88. Define bus arbitration. List out its types. (May / June 2009)

Bus Arbitration: It is the process by which the next device to become the bus master is selected and the bus mastership is transferred to it.

Types: There are 2 approaches to bus arbitration. They are,

- Centralized arbitration (A single bus arbiter performs arbitration)
- Distributed arbitration (all devices participate in the selection of next bus master).

89. Define memory interleaving. (Apr / May 2017)

Memory interleaving is the technique used to increase the throughput. The core idea is to split the memory system into independent banks, which can answer read or write requests independently in parallel.

90. What is an interrupt?

- An interrupt is an event that causes the execution of one program to be suspended and another program to be executed.

91. What are the uses of interrupts?

- Recovery from errors
- Debugging
- Communication between programs
- Use of interrupts in operating system

92. Define vectored interrupts.

- In order to reduce the overhead involved in the polling process, a device requesting an interrupt may identify itself directly to the CPU. Then, the CPU can immediately start executing the corresponding interrupt-service routine. The term vectored interrupts refers to all interrupt handling schemes based on this approach.

93. What are the steps taken when an interrupt occurs?

Source of the interrupt

- The memory address of the required ISP
- The program counter & CPU information saved in subroutine
- Transfer control back to the interrupted program

94. Summarize the sequence of events involved in handling an interrupt request from a single device. (Apr / May 2017) Write the sequence of operations carried out by a processor when interrupted by a peripheral device connected to it. (Apr/May 2018)

Let us summarize the sequence of events involved in handling an interrupt request from a single device.

Assuming that interrupts are enabled, the following is a typical scenario.

1. The device raises an interrupt request.
2. The processor interrupts the program currently being executed.
3. Interrupts are disabled by changing the control bits in the PS (except in the case of edge-triggered interrupts).
4. The device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal.
5. The action requested by the interrupt is performed by the interrupt-service routine.
6. Interrupts are enabled and execution of the interrupted program is resumed.

95. Point out how DMA can improve I/O speed. (May / June 2015)

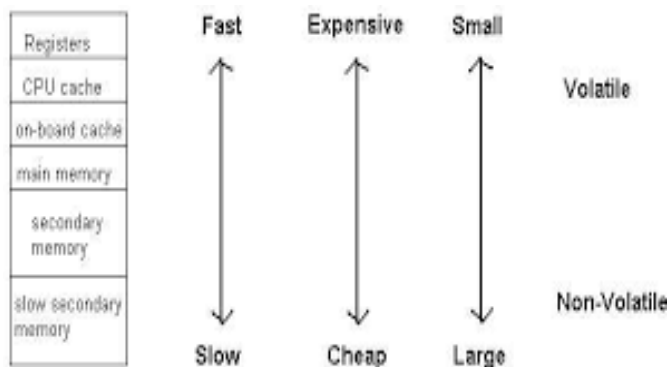
DMA module controls exchange of data between main memory and the I/O device. Because of DMA device can transfer data directly to and from memory, rather than using the CPU as an intermediary, and can thus relieve congestion on the bus. CPU is only involved at the beginning and end of the transfer and interrupted only after entire block has been transferred. In this case DMA improves the I/O speed of the system.

96. Define IO Processor.

The IOP attaches to the system I/O bus and one or more input/output adapters (IOAs). The IOP processes instructions from the system and works with the IOAs to control the I/O devices.

97. Draw the Memory hierarchy in a typical computer system. (Nov/Dec 2018)(Or)

Draw the basic structure of a memory hierarchy. (Apr/May 2019)



98. What is meant by Memory-mapped I/O? (Nov/Dec 2018)

Memory mapped I/O is a way to exchange data and instructions between a CPU and peripheral devices attached to it. **Memory mapped IO** is one where the processor and the IO device share the same **memory location(memory)**, i.e., the processor and IO devices are **mapped** using the **memory** address.

99. What is the use of DMA Controller? (Apr/May 2018)

- A special control unit may be provided to allow the transfer of large block of data at high speed directly between the external device and main memory, without continuous intervention by the processor. This approach is called DMA.
- DMA transfers are performed by a control circuit called the **DMA Controller**.

100. Define Bus Structures.

In computer **architecture**, a **bus** (a contraction of the Latin omnibus) is a communication system that transfers data between components inside a computer, or between computers. This expression covers all related hardware components (wire, optical fiber, etc.) and software, including communication protocols.

101. What is the meaning of USB?(Nov/Dec 2019)

A Universal Serial Bus (**USB**) is a common interface that enables communication between devices and a host controller such as a personal computer (PC). It connects peripheral devices such as digital cameras, mice, keyboards, printers, scanners, media devices, external hard drives and flash drives.

102. what is baud rate?Nov/Dec 2020.

The baud rate is the rate at which information is transferred in a communication channel. Baud rate is commonly used when discussing electronics that use serial communication. In the serial port context, "9600 baud" means that the serial port is capable of transferring a maximum of 9600 bits per second.

At baud rates above 76,800, the cable length will need to be reduced. The higher the baud rate, the more sensitive the cable becomes to the quality of installation, due to how much of the wire is untwisted around each device.

103. In memory organization,what is temporal locality? Nov/Dec 2021

Temporal locality means current data or instruction that is being fetched may be needed soon.

When CPU accesses the current main memory location for reading required data or instruction, it also gets stored in the cache memory which is based on the fact that same data or instruction may be needed in near future.

104. How many total bits are required for a direct-mapped cache with 16KB of data and 4 word blocks, assuming a 32-bit address? (Apr/May 2019)

$$16 \text{ KB} = 16384 (2^{14}) \text{ bytes} = 4096 (2^{12}) \text{ words}$$

$$\text{Block size of } 4 (2^2) \text{ words} = 16 \text{ bytes } (2^4) = 1024$$

$$(2^{10}) \text{ blocks with } 4 \times 32 = 128 \text{ bits of data}$$

$$\text{So, } n = 10$$

$$m = 2$$

$$= 2^{10} \times (4 \times 32 + (32 - 10 - 2 - 2) + 1)$$

$$= 2^{10} \times 147$$

$$= 18.4 \text{ KB}$$

105. Define SATA.

Serial ATA is a peripheral interface created in 2003 to replace Parallel ATA, also known as IDE. Hard drive speeds were getting faster, and would soon outpace the capabilities of the older standard—the

fastest PATA speed achieved was 133MB/s, while SATA began at 150MB/s and was designed with future performance in mind.

106. Explain about USB.

- The **universal serial bus (USB)** is a standard interface for connecting a wide range of devices to the computer such as keyboard, mouse, smart phones, speakers, cameras etc.
- The USB was introduced for commercial use in the year 1995 at that time it has a data transfer speed of 12 megabits/s.